Embedded and MCUs | Volume 16

We get technical

How to implement a voice user interface on resource-constrained MCUs

How single-board computers extend the reach of industrial automation

A guide for the ESP32 microcontroller series

How to perform firmware updates without halting firmware execution

DigiKey

contents

4 How to implement a voice user interface on resource-constrained MCUs

10 The co-processor architecture: an embedded system architecture for rapid prototyping

20 How single-board computers extend the reach of industrial automation

24 Getting started with the Raspberry Pi Pico multicore microcontroller board using C

28 A guide for the ESP32 microcontroller series

34 **Special feature: retroelectro** The birth of the microprocessor and Chuck Peddle

44 How to select and use an audio codec and microcontroller for embedded audio feedback files

50 How to perform firmware updates without halting firmware execution

54 How to implement Time Sensitive Networking to ensure deterministic communication

Editor's note

Embedded systems and microcontrollers (MCUs) form the backbone of modern electronics, driving innovation across industries from industrial automation and automotive to consumer electronics and IoT. As embedded technology evolves, engineers must navigate a landscape of increasing complexity, balancing power efficiency, performance, and security while meeting the growing demands for connectivity and real-time processing.

One of the biggest shifts in embedded development today is the integration of AI and machine learning at the edge. The rise of MCUs with dedicated AI acceleration is enabling real-time inferencing in applications such as predictive maintenance, machine vision, and intelligent automation. Engineers now have access to hardware platforms that can perform complex computations locally, reducing the need for cloud reliance while improving latency and security.

Another key trend is the ongoing push towards RISC-V architecture. As an open-source alternative to proprietary instruction set architectures, RISC-V is gaining traction among developers looking for customization, scalability, and cost-effective solutions. This shift is reshaping the embedded industry, offering engineers new opportunities to design tailored solutions while fostering innovation through open collaboration.

Security remains a critical concern, particularly as more embedded systems become connected. From secure boot and hardware root of trust to post-quantum cryptography, the demand for robust security frameworks is growing. Engineers must consider not just performance and efficiency but also long-term resilience against emerging cybersecurity threats.

This ebook explores the latest advancements in embedded systems and MCUs, offering insights into how engineers can harness cutting-edge technology to develop next-generation applications.

How to implement a voice user interface on resource-constrained MCUs

Written by DigiKey's North American Editors

Smart speakers and other connected hubs form the heart of the smart home, allowing users to control devices and access the Internet. Two trends are apparent as these devices proliferate: users prefer voice control over button presses or complicated menu systems, and there is increasing discomfort with continuous Cloud connectivity because of privacy concerns.

However, a robust and secure voice user interface (VUI) typically demands powerful hardware and complex software for voice recognition. Anything less will likely result in poor performance and unsatisfactory user experiences. Also, many smart speakers and hubs are battery powered, so a VUI must be achieved within a tight power budget. Such an ambitious project can be daunting for a developer lacking experience with voice interfaces.

Chip makers are responding by introducing a technique based on phonemes that significantly reduces the processing requirements. The result is highly accurate and efficient VUI software that can run on familiar 32-bit microcontrollers (MCUs) and is supported by easy-to-use design tools.

This article describes VUI challenges and use cases. It then introduces commercial, easy-to-use MCU application software and local phoneme-based VUI software for connected home applications. The article concludes by showing developers how to get started on VUI projects using Renesas MCUs, VUI software, and evaluation kits.

The challenges of building a VUI

A VUI is speech recognition technology that enables interaction with a computer, smartphone, home automation system, or other device using voice commands. After early engineering challenges, the technology has matured into a reliable control interface and is now widely used in smart speakers and other smart home devices. The key benefit of a VUI is its convenience: instant control from anywhere within voice range with no need to use a keyboard, mouse, buttons, menus, or other interfaces to input commands (Figure 1).

Figure 1: VUI technology has been widely adopted in homes and smart buildings because it is convenient and flexible. Image source: Renesas

The downside of a VUI is its complexity. Conventional technology is based on the lengthy training of a model with specific words or phrases. But natural language processing is word-order independent, which demands considerable development work and significant computing power to run in real-time. This has slowed the broader adoption of VUIs.

Now, a new technique simplifies VUI software to the extent that it can run on small, efficient microcontrollers (MCUs) such as Arm Cortex-M devices. This technique relies on the fact that all words in each spoken language are made up of linguistic sounds called phonemes. There are far fewer phonemes than words; English has 44, Italian has 32, and the traditional Hawaiian language

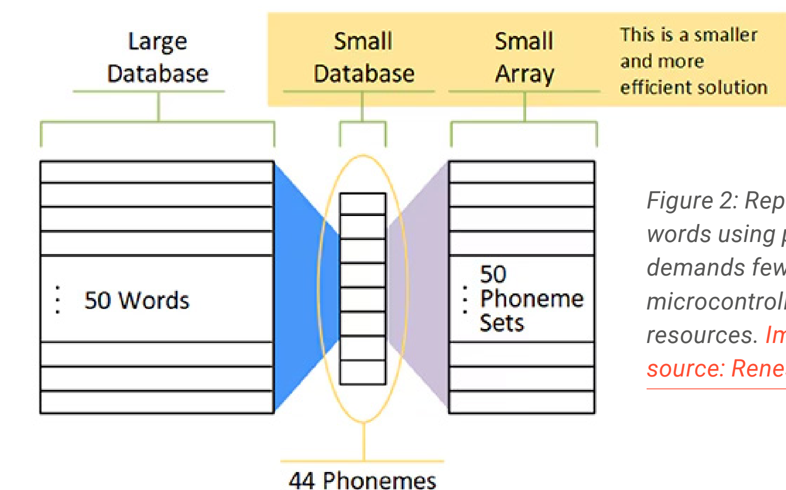


Figure 2: Representing words using phonemes demands fewer microcontroller resources. Image source: Renesas

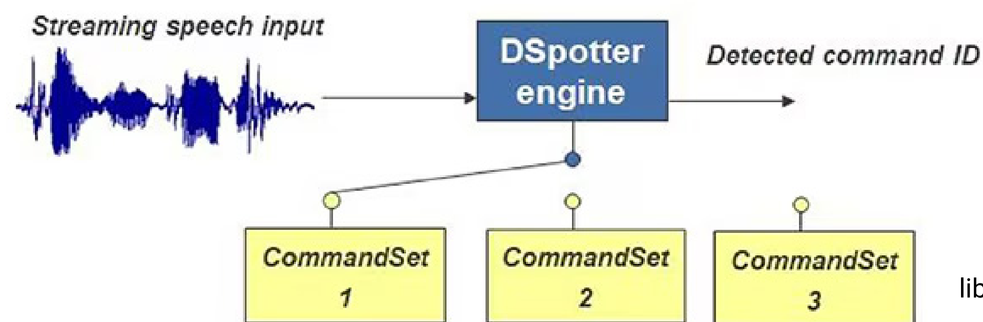


Figure 3: The DSpotter tool allows the creation of 'CommandSets' that can be logically connected by the developer's program to create a VUI with different levels. Image source: Renesas

has just 14. If a VUI uses an English command set of 200 words, each word could be broken down into its associated phonemes from the set of 44.

Within VUI software, each phoneme could then be identified by a numeric code (or a 'token'), with the various tokens forming the language. Storing words as sounds requires extensive computational resources and takes up far more memory space than phonemes stored as tokens. Processing phoneme tokens (and thus command words) in an expected order further simplifies computation and makes it possible to run VUI software locally on a modest MCU (Figure 2).

This means that the software efficiencies achieved by using phonemes allow the processing to run locally. Removing the need for Cloud processing means there is no requirement for continuous internet connectivity that introduces user privacy and data security concerns.

Renesas has shown a commercial VUI software package based on the phoneme principle as part of its ecosystem. The software, called Cyberon DSpotter, creates a VUI algorithm that is streamlined enough to run on Renesas RA series MCUs featuring Arm Cortex-M4 and M33 cores.

Developing with Cyberon DSpotter

Cyberon DSpotter is built on a library of phonemes and phoneme combinations. This is an alternative approach to the traditional and computing-heavy training of algorithms to recognize specific words. To break down words into phonemes and then represent them as tokens, the developer can use the DSpotter Modeling Tool.

DSpotter is embedded (non-Cloud) software that works as a local voice trigger and command-recognition solution with robust noise reduction. It consumes minimal resources and is highly accurate. Depending on the selected MCU, secure data transfer can also be implemented.

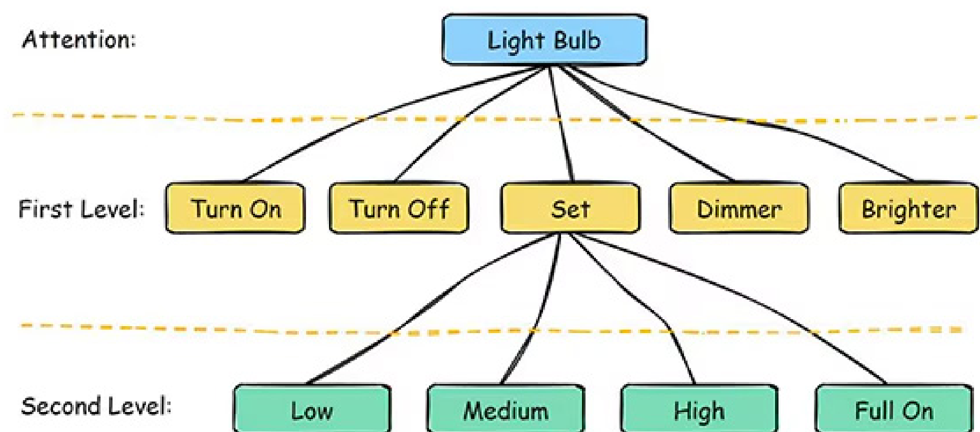


Figure 4: The streamlined nature of Cyberon DSpotter requires that commands follow a logical sequence, or they won't be recognized. Image source: Renesas

DSpotter asks for each command word or phrase, which the tool breaks down into phonemes. The command set and supporting data for the VUI are then built into a binary file that the developer includes in the project along with the Cyberon library. The library and the binary file are used together on the MCU to support the recognition of the desired speech commands.

The DSpotter tool creates 'CommandSets' that can be logically connected by the developer's program to create a VUI with different levels. This allows for multi-level commands such as, 'I'd like the lightbulb set to high, please': the command words being 'lightbulb', followed by 'set', and 'high'. Each command in a group has its own index, as does each command within a level (Figure 3).

The DSpotter library processes incoming sound and searches for phonemes that match the commands in the database. When it finds a match, it returns with the index and group numbers. Such an arrangement allows the main application code to create a hierarchical switch statement to process the command words/phrases as they come. The resulting library can be small enough to fit on an MCU with just 256 kilobytes (Kbytes) of flash memory and 32 Kbytes of SRAM. The CommandSet can grow if more memory is available.

It is important for the developer

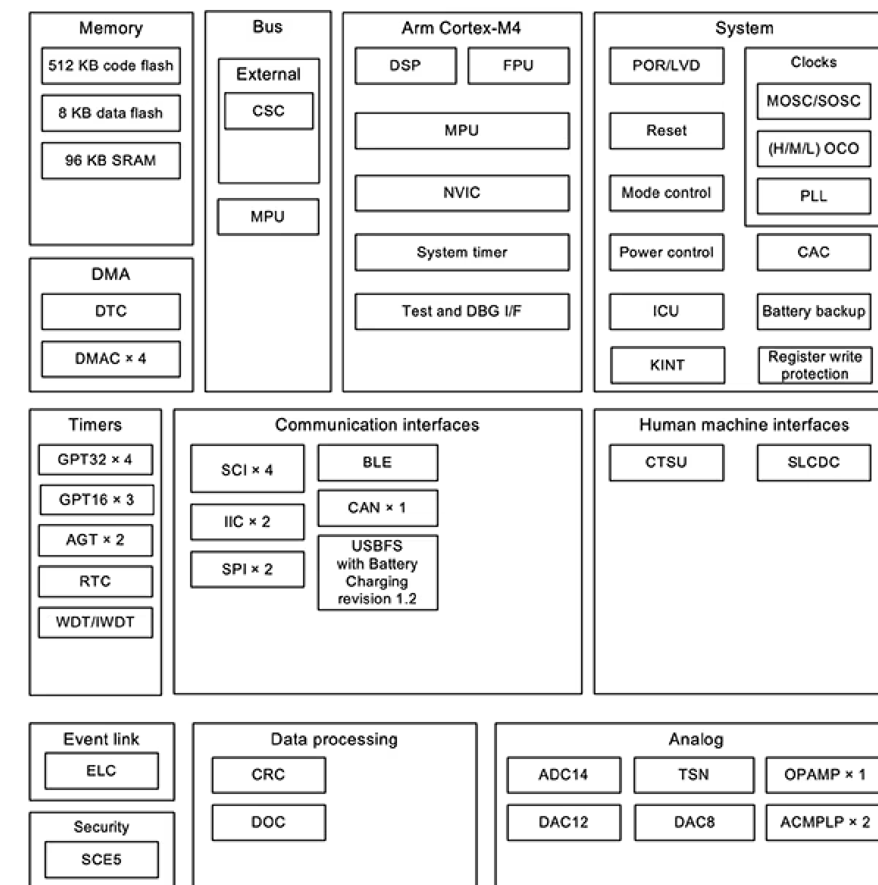


Figure 5: The R7FA4W1AD2CNG MCU provides ample resources to build a non-Cloud VUI for applications like a smart light switch. Image source: Renesas

to appreciate that there are limitations to the phoneme method for a VUI. The relatively limited resources of the MCU dictate that Cyberon DSpotter is speech recognition rather than voice recognition. This means the software cannot perform natural language processing. Hence, if the command words don't follow a logical sequence (for example, 'high', 'lightbulb', 'set' instead of 'lightbulb', 'set', 'high'), the system won't recognize the command and will reset back to the top level.

One design suggestion is to add a visual indicator to the VUI (for example, an LED) to indicate when the processor assumes it is at the top level of the CommandSet, prompting the user to reissue the command in the logical sequence (Figure 4).

Running a non-Cloud VUI with restricted resources

The efficiency of Cyberon DSpotter allows it to run on Renesas' RA2, RA4, and RA6 families of Arm Cortex-M MCUs. These are popular

across a wide range of consumer, industrial, and IoT applications. They are supported by easy-to-use design tools, making it relatively straightforward to build a simple VUI without extensive coding experience or in-house expertise.

The choice of a particular RA family MCU primarily comes down to the complexity of commands and the Cyberon library's size. A smart light switch, which requires a modest command set and limited computing power to operate effectively, could be based on the R7FA4W1AD2CNG from the RA4 family. This MCU has a battery-friendly 48-megahertz (MHz) Arm Cortex-M4 core supported by 512 Kbytes of flash memory and 96 Kbytes of SRAM. It features a segment LCD controller, a capacitive touch sensing unit, Bluetooth Low Energy (Bluetooth LE) wireless connectivity, USB 2.0 Full-Speed, a 14-bit analog-to-digital converter (ADC), a 12-bit digital-to-analog converter (DAC), plus security and safety features (Figure 5).

A more extensive Cyberon DSpotter library and a more powerful core are needed for an application such as a smart speaker. A suitable candidate is the R7FA6M4AF3CFM. This MCU from the RA6 family features the more powerful 200 MHz Arm Cortex-M33 core supported by 1 megabyte (Mbyte) of flash memory and 256 Kbytes of SRAM. It has a CAN bus, Ethernet, I²C, LIN bus, a capacitive touch sensing unit, and many other interfaces and peripherals.

The RA4 and RA6 families are supported by evaluation boards, the RTK7EKA4W1S00000BJ and the RTK7EKA6M4S00001BE, respectively, to allow a developer to exercise the MCUs' capabilities. Each evaluation board has the target MCU and an onboard debugger.

Renesas also offers a VUI solution kit to accelerate development. The kit is similar to the evaluation boards in that it incorporates the target device and debuggers. The board also features several I/O interfaces and has four microphones: two analog and two digital.

Access to the software needed for development with the VUI solution kit is available on Cyberon's website. This includes complimentary Cyberon DSpotter Modeling Tool access and features an e2 studio project with a working voice CommandSet (e2 studio is an Eclipse-based integrated development environment (IDE) for Renesas MCUs). The example CommandSet can be used as a template for developing custom voice command sequences. The system's reactions can then be monitored using a terminal window. It generally takes about 15 minutes to create the VUI structure shown in Figure 4.

More sophisticated application software design for the Cyberon package is supported by the company's Renesas Flexible Software Package (FSP) for embedded system designs using the RA families. The FSP is based on an open software ecosystem and includes Azure RTOS or FreeRTOS, legacy code, and third-party ecosystems. It can run in several IDEs, including e2 studio.

It is one thing for a VUI to perform well in a quiet laboratory, but quite another for it to work accurately with significant background noise.

How well does the VUI perform?

It is one thing for a VUI to perform well in a quiet laboratory, but quite another for it to work accurately with significant background noise. A typical operating environment for a smart speaker could include a TV or radio, conversation, other music sources, and the general hubbub of a household or a social gathering. Moreover, the VUI will have to contend with dialects and less-than-perfect diction. Despite these challenges, users expect almost flawless performance.

SNR	Background Noise	Distance	Hit-rate	Alexa Requirements
(Clean)	none	1.5 m	100.00%	90%
(Clean)	none	3 m	100.00%	90%
10 dB	Babble	1.5 m	98.55%	80%
10 dB	Babble	3 m	98.84%	80%
10 dB	Music	1.5 m	98.26%	80%
10 dB	Music	3 m	98.55%	80%
10 dB	TV	1.5 m	98.84%	80%
10 dB	TV	3 m	98.55%	80%
5 dB	Babble	1.5 m	98.84%	80%
5 dB	Babble	3 m	96.24%	80%
5 dB	Music	1.5 m	98.84%	80%
5 dB	Music	3 m	97.08%	80%
5 dB	TV	1.5 m	93.37%	80%
5 dB	TV	3 m	90.72%	80%

Table 1: Command success test results for a Cyberon-powered VUI with various sources of background noise. In all cases, the VUI outperformed the Amazon Alexa benchmark. Image source: Renesas

To improve performance in a difficult listening environment, Cyberon DSpotter software running on the Renesas RA family of MCUs includes noise immunity features that require minimal processor resources. To demonstrate its efficacy, tests were done with a Cyberon DSpotter VUI listening to commands while subject to various background noise sources at 1.5- and 3-meter (m) distances, and with signal-to-noise ratios (SNRs) of 0, 5, and 10 decibels (dB). In all cases, the VUI outperformed the Amazon Alexa benchmark (Table 1).

Conclusion

VUIs are rapidly becoming the preferred consumer control interface for smart products. A speech control approach using phonemes as the basis of commands and a strict command structure can dramatically reduce memory and computing requirements, allowing the technology to run locally on small, resource-constrained MCUs.

The co-processor architecture: an embedded system architecture for rapid prototyping

Written By Noah Madinger, Colorado Electronic Product Design (CEPD)

Editor's note: although well known for its digital processing performance and throughput, the co-processor architecture provides the embedded systems designer opportunities to implement project management strategies, which improve both development costs and time to market. This article, focused specifically upon the combination of a discrete microcontroller (MCU) and a discrete field programmable gate array (FPGA), showcases how this architecture lends itself to an efficient and iterative design process. Leveraging researched sources, empirical findings, and case studies, the benefits of this architecture are explored, and exemplary applications are provided. Upon this article's conclusion, the embedded systems designer will have a better understanding of when and how to implement this versatile hardware architecture.

Introduction

The embedded systems designer finds themselves at a juncture of design constraints, performance expectations, and schedule and budgetary concerns. Indeed, even the contradictions in modern project management buzzwords and phrases further underscore the precarious nature of this role: 'fail fast'; 'be agile'; 'future-proof it'; and 'be disruptive!'. The acrobatics involved in even trying to satisfy these expectations can be harrowing, and yet, they have been spoken and continue to be reinforced throughout the market. What is needed is a design approach, which allows for an evolutionary iterative process to be implemented, and just like with most embedded systems, it begins with the hardware architecture.

The co-processor architecture, a hardware architecture known for combining the strengths of both microcontroller unit (MCU) and field programmable gate array (FPGA) technologies, can offer the embedded designer a process capable of meeting even the most demanding requirements, and yet it allows for the flexibility necessary to address both known and unknown challenges. By providing hardware capable of iteratively adapting, the designer can demonstrate progress, hit critical milestones, and take full advantage of the rapid prototyping process.

Within this process are key project milestones, each with their own unique value to add to the development effort. Throughout this article, these will be referred to by the following terms: The Digital Signal Processing with the Microcontroller milestone, the System Management with the Microcontroller milestone, and the Product Deployment milestone.

By the conclusion of this article, it will be demonstrated that a flexible hardware architecture can be better suited to modern embedded systems design than a more rigid approach. Furthermore, this approach can result in improvements to both project cost and time to market. Arguments, provided examples, and case studies will be used to defend this position. By observing the value of each milestone within the design flexibility that this architecture provides, it becomes clear that an adaptive hardware architecture is a powerful driver in pushing embedded systems design forward.

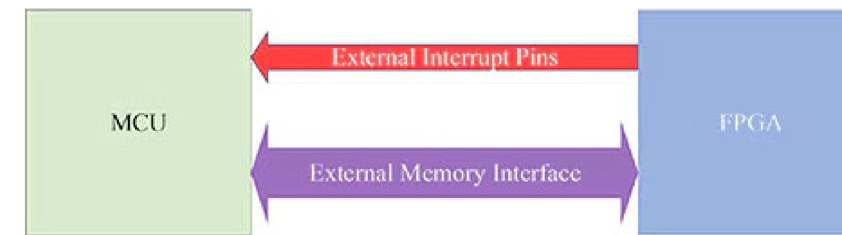


Figure 1: Generic co-processor diagram (MCU + FPGA). Image source: CEPD

Exploring the strengths of the co-processor architecture: design flexibility and high-performance processing

A common application for FPGA designs is to interface directly with a high-speed analog-to-digital converter (ADC). The signal is digitized, read into the FPGA, and then some digital signal processor (DSP) algorithms are applied to this signal. Last of all, the FPGA then makes decisions based upon the findings.

Such an application will serve as the example throughout this article. Furthermore, Figure 1 illustrates a generic co-processor architecture, where the MCU and FPGA are connected through the MCU's external memory interface. The FPGA is treated as if it were a piece of external static random-access memory (SRAM). Signals come back to the MCU from the FPGA and serve as hardware interrupt lines and status indicators. This allows the FPGA to indicate critical states to the MCU, such as communicating that an ADC conversion is ready, or a fault has occurred, or another noteworthy event has happened.

The strengths of the co-processor approach are probably best seen within the deliverables of each of the above-mentioned milestones. Value is assessed by not only listing the accomplishments of a task or phase but also by assessing the enablement that these accomplishments allow. The answers to the following questions assist in assessing the overall value of a milestone's deliverables:

- Can the progress of other team members now more rapidly continue, as project dependencies and bottlenecks are removed?

- How do the accomplishments of the milestone enable further parallel execution paths?

The digital signal processing with the microcontroller milestone

The first development stage that this hardware architecture allows places the MCU front and center. All things being equal, MCU and executable software development is less resource and time-consuming than FPGA and hardware descriptive language (HDL) development. Thus, by initiating product development with the MCU as the primary processor, algorithms can be implemented, tested, and validated more rapidly. This allows algorithmic and logical bugs to be discovered early in the design process, and this also allows for substantial portions of the signal chain to be tested and validated.

The FPGA's role in this initial milestone is to serve as a high-speed data gathering interface. Its task is to reliably pipe data from the high-speed ADC, alert the MCU that data is available, and present this data on the MCU's external memory interface. Although this role does not include implementing HDL-based DSP processes or other algorithms, it is nonetheless highly critical.

The FPGA development performed at this phase lays the foundation for the product's ultimate success both

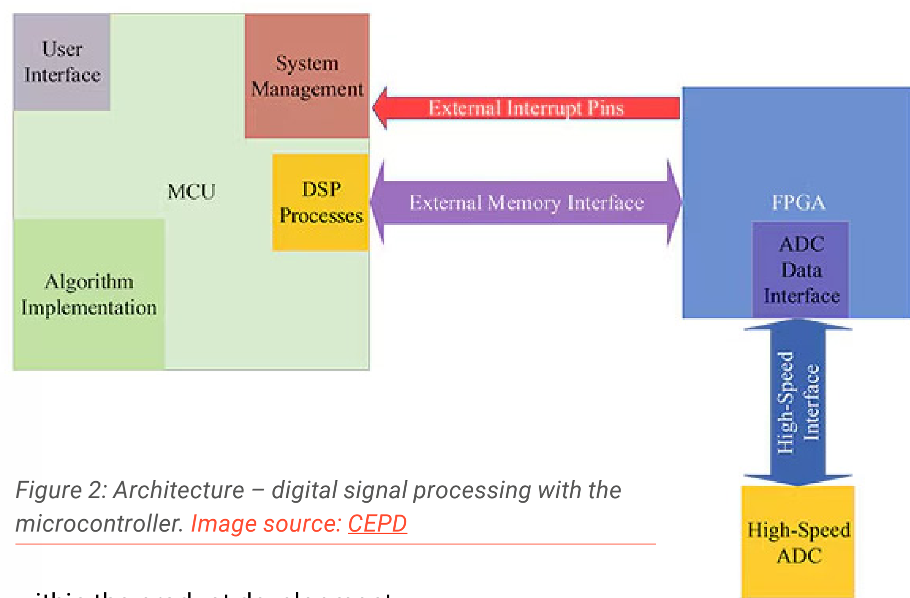


Figure 2: Architecture – digital signal processing with the microcontroller. Image source: CEPD

within the product development efforts and upon release to the market. By focusing on just the low-level interface, adequate time can be dedicated to testing these essential operations. Only once the FPGA is reliably and confidently performing this interfacing role, can this milestone be concluded confidently.

Key deliverables from this initial milestone include the following benefits:

1. The full signal path – all amplifications, attenuations, and conversions – will have been tested and validated
2. The project development time and effort will have been reduced by initially implementing the algorithms in software (C/C++); this is of considerable value to management and other stakeholders, who must see the feasibility of this project before approving future design phases
3. The lessons learned from implementing the algorithms in C/C++ will be directly transferable to HDL implementations – through the use of software-to-HDL tools, e.g., Xilinx HLS

The system management with the microcontroller milestone

The second development stage, which this co-processor approach offers, is defined by the moving of DSP processes and algorithm implementations from the MCU to the FPGA. The FPGA is still responsible for the high-speed ADC interface, however, by assuming these other roles, the speed and parallelism offered by the FPGA are fully utilized. Additionally, unlike the MCU, multiple instances of the DSP processes and algorithm channels can be implemented and run simultaneously.

Built upon the lesson learned from the MCU's implementation, the designer carries this confidence forward into this next milestone. Tools, such as the aforementioned Vivado HLS from Xilinx, provide a functional translation from the executable C/C++ code to synthesizable HDL. Now, timing constraints, process parameters, and other user preferences must still be defined and implemented, however, the core functionality is persevered and translated to the FPGA fabric.

For this milestone, the MCU's role is that of a system manager. Status and control registers within the FPGA are monitored, updated, and reported on by the MCU. Furthermore, the MCU manages the user interface (UI). This UI could take the form of the web server accessed over an Ethernet or Wi-Fi connection, or it could be an industrial touchscreen interface giving access to users at the point

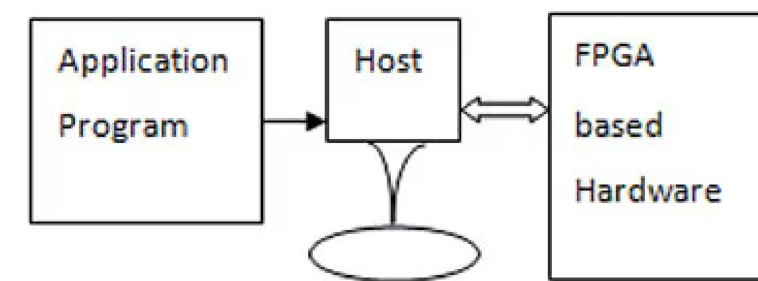


Figure 4: Application program, host processor, and FPGA-based hardware - used in satellite communications example.

of use. The key takeaway from the MCU's new, more refined role is this: by being relieved from the computationally intensive processing tasks, both the MCU and FPGA are now being leveraged in tasks for which they are well suited.

Key deliverables form this milestone and include these benefits:

1. Fast, parallel execution of DSP processes and algorithm implementations are being provided by the FPGA.The MCU provides a responsive and streamlined UI and manages the product's processes
2. Having been first developed and validated within the MCU, algorithmic risks have been mitigated, and these mitigations are translated over into synthesizable HDL. Tools, such as Vivado HLS, make this translation an easier process. Furthermore, FPGA-specific risks can be mitigated through integrated simulation tools, such as the Vivado design suite
3. Stakeholders are not exposed to significant risk by moving the processes over to the FPGA. On the contrary, they get to see and enjoy the benefits that the FPGA's speed and parallelism provide. Measurable performance improvements are observed and focus can now be given to readying this design for manufacturing

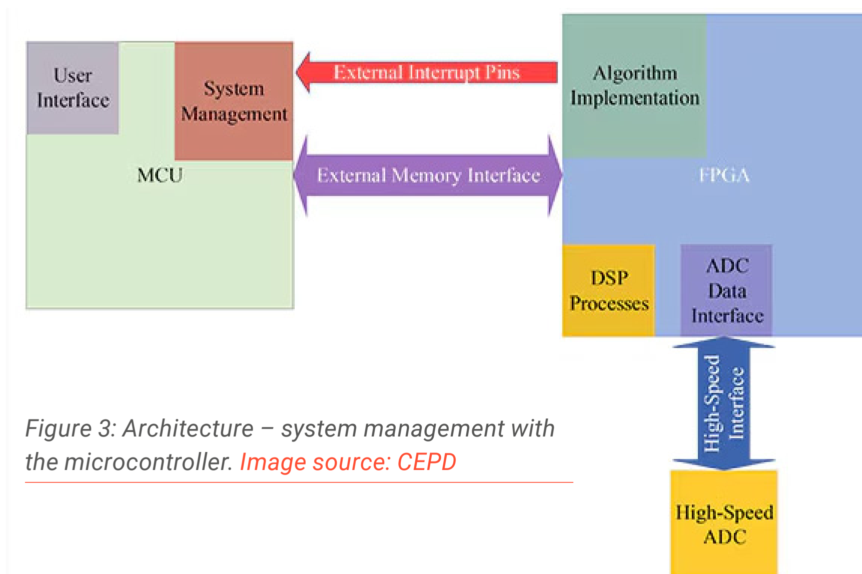


Figure 3: Architecture – system management with the microcontroller. Image source: CEPD

The product deployment milestone

With the computationally intensive processing being addressed within the FPGA, and the MCU handling its system management and user interface roles, the product is ready for deployment. Now, this paper does not advocate for bypassing Alpha and Beta releases; however, the emphasis for this milestone are the capabilities that the co-processor architecture provides to product deployment.

Both the MCU and FPGA are field updateable devices. Several advancements have been made to make FPGA updates just as accessible as software updates. Moreover, since the FPGA is within the addressable memory space of the MCU, the MCU can serve as the access point for the entire system: receiving both updates for itself as well as for the FPGA. Updates can be conditionally scheduled, distributed, and customized on a per end-user basis. Last of all, user and use-case logs can be maintained and associated with specific build implementations. From these data sets, performance can continue to be refined and enhanced even after the product is in the field.

Perhaps the strengths of this total-system updatability are no more underscored than in space-based applications. Once a product is launched, maintenance and updates must be performed remotely. This could be as simple as changing logical conditions, or as complicated as updating a communications modulation scheme. The programmability offered by FPGA technologies and the co-processor architecture can accommodate the entirety of this range of capabilities, all while offering radiation-hardened component choices.

The final key takeaway from this milestone is progressive cost reduction. Cost reductions, bill of materials (BOM) changes, and other optimizations can also occur at this milestone. During field deployments, it may be discovered that the product can operate just as well with a less expensive MCU, or less capable FPGA. Because of the co-processor, architecture designers are not stuck using components whose capabilities exceed their application's needs. Furthermore, should a component become unavailable, the architecture allows for new components to be integrated into the design. This is not the case with a single-chip, system on a chip (SoC) architecture, or with a high-performance DSP or MCU that attempts to handle all of the product's processing. The co-processor architecture is a good mix of capability and flexibility giving the designer more choices and freedoms both with the development phases and upon release to the market.

Supporting research and related case studies

Satellite communications example

In short, the value of a co-processor is to offload the primary processing unit so that tasks are executed upon hardware, in which accelerations and streamlining can be taken advantage of. The advantage of such a design choice is a net increase in computational speed and capabilities, and, as this article argues, a reduction in development cost and development time. Perhaps one of the most compelling realms for these benefits is in the area of space communications systems.

In their publication, FPGA based hardware as coprocessor, G. Prasad and N. Vasantha detail how data processing within an FPGA blends the computational needs of satellite communications systems without the high non-recurring engineering (NRE) costs of application-specific integrated circuits (ASICs) or the application-specific limitations of a hard-architecture processor. Just as was described in the Digital Signal Processing with the Microcontroller Milestone, their design begins with the application processor performing a majority of the computationally intensive algorithms. From this starting point, they identify the key sections of software that consume a majority of the central processing unit (CPU) clock's cycles and migrate these sections over to HDL implementation. The graphical representation is highly similar to what has been presented so far, however, they have chosen to represent the Application Program as its own independent block, as it can be either realized in the Host (Processor) or in the FPGA based Hardware.

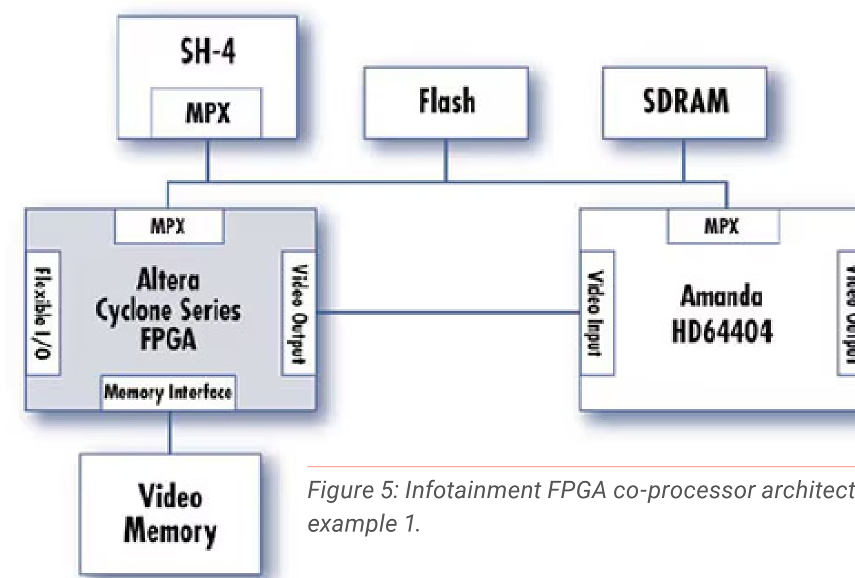


Figure 5: Infotainment FPGA co-processor architecture example 1.

By utilizing a peripheral component interconnect (PCI) interface and the host processor's direct memory access(DMA), peripheral performance is dramatically increased. This is mostly observed within the improvements for the Derandomization process. When this process was performed in the host processor's software, there was clearly a bottleneck in the real-time response of the system. However, when moved to the FPGA, the following benefits were observed:

- The Derandomization process executed in real-time without causing bottlenecks
- The host processor's computational overhead was significantly reduced, and it could now better perform a desired logging role
- The total performance of the entire system was scaled up

All of this was achieved without the costs associated with an ASIC, and while enjoying the flexibility of programmable logic [5]. Satellite communications present considerable challenges, and this approach can verifiably meet these requirements, and continue to provide design flexibility.

Automotive infotainment example

Entertainment systems within automobiles are distinguishing features for discerning consumers. Unlike a majority of automotive electronics, these devices are highly visible and are expected to provide exceptional response time and performance. However, designers are often squeezed between the current needs of the design and the flexibility, which future features will require. For this example, the implementation needs of signal processing and wireless communications will be used to highlight the strengths of the co-processor hardware architecture.

One of the predominant automotive entertainment system architectures used was published by the Delphi Delco Electronics Systems corporation. This architecture employed an SH-4 MCU with a companion ASIC, Hitachi's HD64404 Amanda peripheral. This

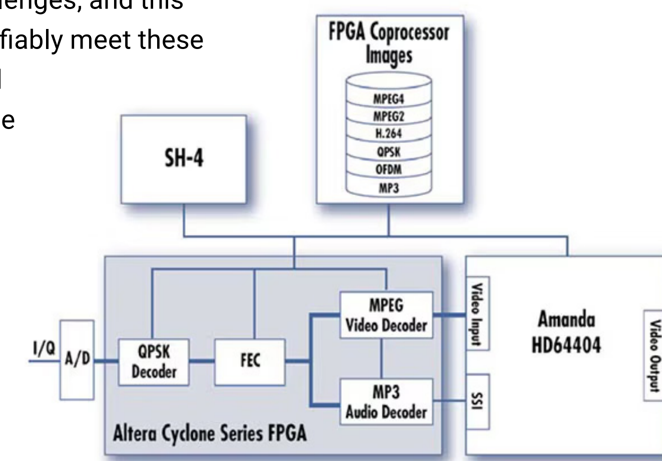


Figure 6: Infotainment FPGA co-processor architecture example 2.

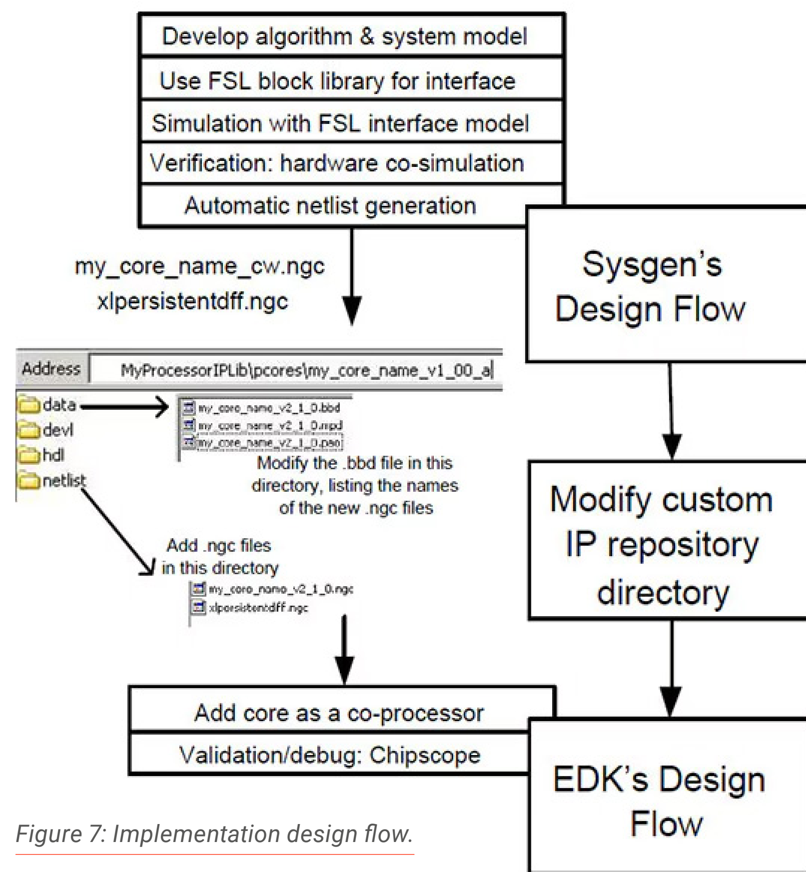


Figure 7: Implementation design flow.

architecture satisfied over 75% of the automotive market's baseline entertainment functionality; however, it lacked the ability to address video processing applications and wireless communications. By including an FPGA within this existing architecture, further flexibility and capability can be added to this already-existing design approach.

The Figure 5 architecture is suitable for both video processing and wireless communications management. By pushing the DSP functionalities to the FPGA, the Amanda processor can serve a system management role and is freed to implement a wireless communications stack. As both the Amanda and FPGA have access to the external memory, data can be rapidly exchanged among the system's processors and components.

The second infotainment in Figure 6 highlights the FPGA's ability to address both the incoming high-speed analog data and the handling of the compression and encoding needed for video applications. In fact, all of this functionality can be pushed into the FPGA and through the use of parallel processing, these can all be addressed in real-time.

By including an FPGA within an existing hardware architecture, the proven performance of the existing hardware can be coupled with flexibility and futureproofing. Even within existing systems, the co-processor architecture provides options to designers, which would otherwise not be available [6].

Rapid prototyping advantages

At its heart, the rapid prototyping process strives to cover a substantial amount of product development area by executing tasks in parallel, identifying 'bugs' and design issues quickly, and validating data and signal paths, especially those within a project's critical path. However, for this process to truly produce streamlined, efficient results there must be sufficient expertise in the project areas required.

Traditionally, this means that there must be a hardware engineer, an embedded software or DSP engineer, and an HDL engineer. Now, there are plenty of interdisciplinary professionals, who may be able to satisfy multiple roles; however, there is still substantial project overhead involved in coordinating these efforts.

In their paper, An FPGA based rapid prototyping platform for wavelet coprocessors, the authors promote the idea that using a co-processor architecture allows a single DSP engineer to fulfil all of these roles, efficiently and effectively. For this study, the team began designing and simulating the desired DSP functionality within MATLAB's Simulink tool. This served two primary functions, in that it, 1) verified the desired performance through simulation, and 2) served as a baseline to which future design choices could be compared and referenced.

After simulation, critical functionalities were identified and divided into different cores – these are soft-core components and processors that can be synthesized within an FPGA. The most important step during this work was to define the interface among these cores and components and to compare the data-exchange performance against the desired, simulated performance. This design process closely aligned with Xilinx's design flow for embedded systems and is summarized in Figure 7.

By dividing the system into synthesizable cores, the DSP engineer can focus upon the most critical aspects of the signal processing chain. She/he does not need to be an expert in hardware or HDL to modify, route, or implement different soft-core processors or components within the FPGA. So long as the designer is aware of the interface and the formats of the data, they have full control over the signal paths and can refine the system's performance.

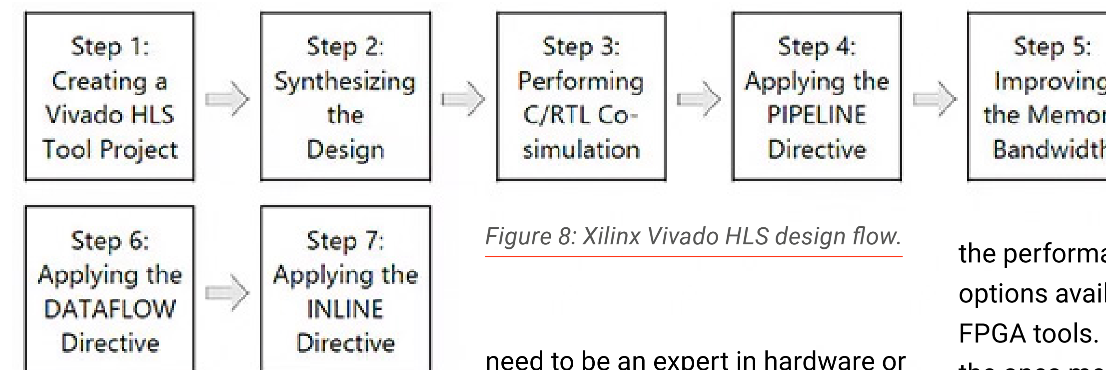


Figure 8: Xilinx Vivado HLS design flow.

Empirical findings – the discrete cosine transform case study

The empirical findings not only confirmed the flexibility availed by the co-processor architecture to the embedded systems designer but also showcased the performance-enhancing options available with modern FPGA tools. Enhancements, like the ones mentioned below, may not be available or may have less impact for other hardware architectures. The discrete cosine transform (DCT) was selected as a computationally intensive algorithm, and its progression from a C-based implementation to an HDL-based implementation was at the heart of these findings. DCT was chosen since this algorithm is used in digital signal processing for pattern recognition and filtering [8]. The empirical findings were based upon a laboratory exercise, which was completed by the author

	Latency		Interval	
	min	max	min	max
Default (solution 1)	2935	2935	2935	2935
Pipeline inner loop (solution 2)	1723	1723	1723	1723
Pipeline outer loop (solution 3)	843	843	843	843
Array partition (solution 4)	477	477	477	477
Dataflow (solution 5)	476	476	343	343
Inline (solution 6)	463	463	98	98

Table 1: FPGA algorithm execution optimization findings (latency and interval).

and coworkers, to obtain the Xilinx Alliance Partner certification for 2020-2021.

The following tools and devices were used in this effort:

- Vivado HLS v2019
- The device for assessment and simulation was the xczu7ev-ffvc1156-2-e

Beginning with the C-based implementation, the DCT algorithm accepts two arrays of 16-bit numbers; array 'a' is the input array to the DCT, and array 'b' is the output array from the DCT. The data width (DW) is therefore defined as 16, and the number of elements within the arrays (N) is 1024/DW, or 64. Last of all, the size of the DCT matrix (DCT_SIZE) is set to 8, which means an 8 x 8 matrix is used.

Following the premise of this article, the C-based algorithm implementation allows the designer to quickly develop and validate the algorithm's functionality. Although it is an important consideration, this validation places functionality at a higher weighting than execution time. This weighting is allowed, since the ultimate implementation of this algorithm will be in an FPGA, where hardware acceleration, loop unrolling, and other techniques are readily available.

Once the DCT code was created within the Vivado HLS tool as a project, the next step is to begin synthesizing the design for FPGA implementation. It is at this next step where some of the most impactful benefits from moving an algorithm's execution from an MCU to an FPGA become more apparent – as a reference this step is equivalent to the System Management with the Microcontroller milestone discussed above.

Modern FPGA tools allow for a suite of optimizations and enhancements that greatly enhance the performance of complex algorithms. Before analyzing the results, there are some important terms to keep in mind:

- Latency – The number of clock cycles required to execute all iterations of the loop [10]
- Interval – The number of clock cycles before the next iteration of a loop starts to process data [11]
- BRAM – Block Random Access Memory
- DSP48E – Digital Signal Processing Slice for the UltraScale Architecture
- FF – Flipflop
- LUT – Look-up Table
- URAM – Unified Random-Access Memory (can be composed of a single transistor)

The co-processor hardware architecture provides the embedded designer with a high-performance platform that maintains its design flexibility throughout development and past product release.

Default

The default optimization setting comes from the unaltered result of translating the C-based algorithm to synthesizable HDL. No optimizations are enabled, and this can be used as a performance reference to better understand the other optimizations.

Pipeline inner loop

The PIPELINE directive instructs Vivado HLS to unroll the inner loops so that new data can start being processed while existing data is still in the pipeline. Thus, new data does not have to wait for the existing data to be complete before processing can begin.

Pipeline outer loop

By applying the PIPELINE directive to the outer loop, the outer loop's operations are now pipelined. However, the inner loops' operations now occur concurrently. Both the latency and interval time are cut in half through applying this directly to the outer loop.

Array partition

This directive maps the contents of the loops to arrays and thus flattens all of the memory access to single elements within these arrays. By doing this, more RAM is consumed, but again, the execution time of this algorithm is cut in half.

Dataflow

This directive allows the designer to specify the target number of clock cycles between each of the input reads. This directive is only supported for top-level function. Only loops and functions exposed to this level will benefit from this directive.

Inline

The INLINE directive flattens all loops, both inner and outer. Both row and column processes can now execute concurrently. The number of required clock cycles is kept to a minimum, even if this consumes more FPGA resources.

Conclusion

The co-processor hardware architecture provides the embedded designer with a high-performance platform that maintains its design flexibility throughout development and past product release. By first validating algorithms in C or C++, processes, data and signal paths, and critical functionality can be verified in a relatively short amount of time. Then, by translating the processor-intensive algorithms into the co-processor FPGA, the designer can enjoy the benefits of hardware acceleration and a more modular design.

Should parts become obsolete, or optimizations be required, the same architecture can allow for these changes. New MCUs and new FPGAs can be fitted into the design, all the while the interfaces can remain relatively untouched. Additionally, since both the MCU and FPGA are field updatable, user-specific changes and optimizations can be applied in the field and remotely.

In closing, this architecture blends the development speed and availability of an MCU with the performance and expandability of an FPGA. With optimizations and performance enhancements available at every development step, the co-processor architecture can meet the needs of even the most challenging requirements – both for today's designs and beyond.

Table 2: FPGA algorithm execution optimization findings (resource utilization).

	BRAM_18K	DSP48E	FF	LUT	URAM
Default (solution)	5	1	246	964	0
Pipeline inner loop (solution 2)	5	1	223	1211	0
Pipeline outer loop (solution 3)	5	8	516	1356	0
Array partition (solution 4)	3	8	862	1879	0
Dataflow (solution 5)	3	8	868	1654	0
Inline (solution 6)	3	16	1086	1462	0

How single-board computers extend the reach of industrial automation

Written by Jeff Shepard

The availability of single-board-computers (SBCs) like Arduino and Raspberry Pi, rated for use in industrial environments together with software development tools based on the International Electrotechnical Commission (IEC) 61131-3 standard, have opened new opportunities for machine and factory automation designers. Some of these new SBC-based solutions also open new possibilities for automating environmental monitoring, smart home and building installations, agricultural applications, and other non-industrial systems.

Industrial SBCs are being used in machine controllers, industrial PCs (IPCs), Industrial Internet of Things (IIoT) gateways, micro programmable logic controllers (PLCs), soft PLCs, analog and digital input/output (I/O) modules, and more. These SBC-based devices are built on open hardware and open software platforms, sometimes including full root rights.

Compliance with IEC 61131-3 means that the five standard automation programming languages are supported, including ladder diagram, structured text, function block diagram, sequential function diagram, and instruction list. Being built using SBCs means developers can also turn to languages like Java, Python, C, or C++, providing greater flexibility than traditional industrial control hardware. Some support data security from the hardware to the Cloud or a higher-level network like an enterprise resource planning (ERP) system with an onboard secure element and International Telecommunications Union (ITU) X.509 Standard public key compliance.

This article presents examples of SBC-based solutions available to machine and automation designers from Arduino, Industrial Shields, and KUNBUS for various applications, including small- to medium-scale automation, embedded control in small machines, and large factory automation installations. The article closes with a look at how PROFINET and deterministic networking can be implemented on SBC PLCs.

Arduino PLCs

One of the benefits of most Arduino-based PLCs is the availability of the Arduino PLC integrated development environment (IDE) for writing control software. The Arduino PLC IDE enables users to choose any of the five programming languages defined by IEC 61131-3 and quickly code PLC applications or port existing ones. It also includes ready-to-use Arduino sketches (programs), tutorials, and libraries.

Industrial Shields' Arduino-based PLCs can be programmed using the Arduino IDE or directly using C. These PLCs include open-source tools and can be programmed with multiple software platforms. They can be programmed through the USB or Ethernet ports for remote connections. Users can continuously monitor the status of all the variables, inputs, and outputs.

The model IS.MDUINO.21+ from Industrial Shields is rated for operation from 0°C to +60°C, and its ATmega processor achieves a throughput of 16 MIPS at 16 MHz (Figure 1). Features include:

- 13 Inputs:
  - 7 opto isolated digital (5 VDC to 24 VDC)
  - 2 Interrupts (5 VDC to 24 VDC)
  - 6 software configurable as analog (0 VDC to 10 VDC, 10 bit) or digital (5 VDC to 24 VDC)
- 8 Outputs:
  - 5 opto isolated digital (5 VDC to 24 VDC)
  - 3 software configurable as analog (0 VDC to 10 VDC, 8 bit), digital (5 VDC to 24 VDC), or pulse width modulated (5 VDC to 24 VDC)
- 256 KB memory
- Ethernet, RS-232, RS-485 and USB communications
- Expandable with up to 127 modules

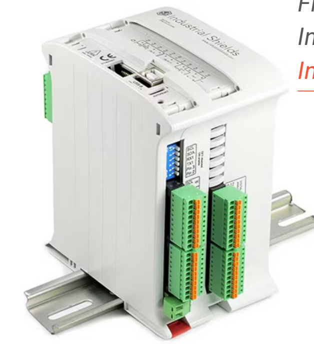


Figure 1: The model IS.MDUINO.21+ from Industrial Shields has 13 inputs and 8 outputs. Image source: Industrial Shields

Micro PLCs

The Arduino Opta is a micro PLC designed to support IIoT applications. Programmable with the Arduino PLC IDE, it supports Arduino sketches and standard PLC languages. The main processor is the dual-core STM32H747 with a 480 MHz Cortex M7, a 240 MHz Cortex M4, and 1 MB program memory that supports real-time control, monitoring, and implementation of predictive maintenance algorithms. Secure over-the-air (OTA) firmware updates are supported by the onboard secure element and X.509 compliance.

Opta PLCs are available in three variants differentiated by their communications capabilities. All three include USB-C. The models are:

- Opta Lite, model AFX00003, that adds 10/100BASE-T Ethernet
- Opta RS485, model AFX00001, that adds 10/100BASE-T Ethernet and half-duplex RS-485
- Opta Wi-Fi, model AFX00002, that adds 10/100BASE-T Ethernet, half-duplex RS-485 802.11 b/g/n Wi-Fi, and Bluetooth low energy (BLE)

These micro PLCs have eight programmable analog/digital inputs and four normally-open relay outputs rated for 10 A (2.3 kW). The real-time clock (RTC) has a typical ten days of power retention at +25°C, and network time protocol (NTP) synchronization is available through the Ethernet port. They are DIN rail compatible to speed system integration (Figure 2).

Embedded PLC for small machines

Designers of small machines for labelling, forming, and sealing, carton packing, gluing, electric ovens, industrial washers and dryers, mixers, and so on can turn to the 170 x 90 x 50 millimeters (mm) Portenta machine control PLC. It has a DIN bar compatible housing and push-in terminals for fast connection and is rated for operation from -40°C to +85°C without external cooling (Figure 3). The main processor is the dual-core STM32H747 with a 480 MHz Cortex M7 and a 240 MHz Cortex M4. The board can support flat screen displays, touch panels, keyboards, joysticks, and mice for installer and operator interfaces. It can be programmed using the Arduino PLC IDE or other embedded development platforms.

The Portenta Machine Control can support predictive maintenance and artificial intelligence (AI) software. Its embedded RTC supports synchronization of processes and enables real-time data collection and remote control of equipment.

Figure 2: Opta Lite Arduino micro PLC showing the four 10 A relay outputs on the left front of the unit. Image source: Arduino

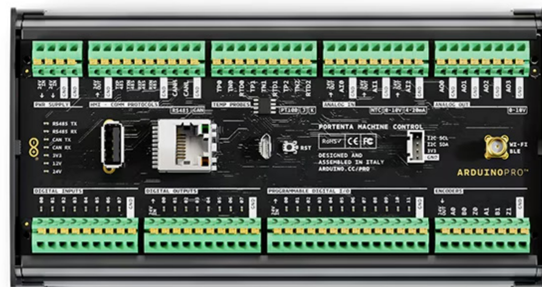


Figure 3: The Portenta Machine Control board is designed for embedded applications in a wide range of machines. Image source: Arduino

It can connect to various external sensors and actuators with isolated and programmable digital and analog I/O connections, three configuration temperature channels, and an I2C connector. Resettable fuses protect all I/Os. Network connectivity is supported by USB, Ethernet, Wi-Fi, BLE and RS-485.

Raspberry Pi for factory automation

More complex automation tasks can benefit from the processing power of Raspberry Pi 4-based PLCs using the Broadcom BCM2711B0 processor. Fabricated on a 28 nanometer (nm) process, the BCM2711B0 uses the Cortex-A72 architecture. It has four cores with a clock speed of 1.5 GHz and 4 GB RAM. It integrates numerous peripherals, including timers, interrupt controller, general purpose I/O (GPIO), USB, PCM/I2S digital audio interface, direct memory access (DMA) controller, I2C masters, serial peripheral interface (SPI) masters, PWM, universal asynchronous receivers/transmitters (UARTs), dual micro HDMI ports that support 4K output, and more.

Industrial Shields' Raspberry Pi Ethernet PLCs use the BCM2711B0, operate with 12 VDC to 24 VDC input voltages, and draw up to 1.5 A of current. They include the Linux operating system and have dual Ethernet ports, dual RS-485 ports, Wi-Fi, BLE, and CAN bus options, making them capable of connecting with many devices using multiple protocols and communications ports. They have been optimized for applications that benefit from real-time control and are available with 2, 4, and 8 GB of RAM. Examples of Industrial Shields' Raspberry Pi PLCs include:

- 012003000200, with 4 GB RAM and 21 I/Os a (Figure 4)
- 012003001100, with 4 GB RAM and 54 I/Os
- 016003000200, with 4 GB RAM, 21 I/Os, and general packet radio service (GPRS) cellular connectivity

Bridging Arduino and Raspberry Pi in PLCs with SimpleComm

The SimpleComm C++ library lets designers send data using RS-485, RS-482, Ethernet, and other protocols. It can be adapted to different communications topologies like ad-hoc, master-slave, and client-server. The original program has an intuitive application programming interface (API) for Arduino environments. Industrial Shields recently adapted SimpleComm for the Linux environment found on Raspberry Pi PLCs.

IPC and IIoT gateway solution

When greater flexibility is needed, designers can turn to KUNBUS' RevPi Core S and SE IPCs and the RevPi Connect S and SE IIoT gateway, all based on Raspberry Pi and designed for DIN rail mounting (Figure 5). In addition to providing circuit diagrams, KUNBUS uses an open-source adaptation of the Raspberry Pi operating system (OS) with a real-time operation patch. The Raspberry Pi OS offers robust interoperability with a wide range of software applications developed for Raspberry Pi. KUNBUS works with software vendors to support supervisory control and data acquisition (SCADA) software for controlling, monitoring, and analyzing industrial devices and processes. The availability of full root access speeds up the implementation of custom programs.

Figure 4: Industrial Shields' Raspberry Pi Ethernet PLC with 4 GB RAM and 21 I/Os. Image source: Industrial Shields

Figure 5: Examples of RevPi Core SE IPC (left) and RevPi Connect IIoT Gateway (right). Image source: KUNBUS

The RevPi Core S and SE are built on an open hardware and open software platform that conforms to the IEC 61131 standard. RevPi Core S units are compatible with all KUNBUS expansion modules, including fieldbus gateways. RevPi Core SE units are compatible with KUNBUS I/O modules but don't support the fieldbus gateways. RevPi Core S/SE IPCs have USB, Micro-USB, Ethernet, and HDMI connections. They feature a 1.5 GHz quad-core processor with 1 GB RAM, and models are available with 8, 16, and 32 GB of storage. For example, model PR100360, RevPi Core S has 16 GB of memory.

To support IIoT connectivity, the RevPi Connect S and SE Gateways are available with up to 32 GB of memory and include two RJ45 10/100 Ethernet sockets, two USB ports, a 4-pin RS-485 interface, plus micro-HDMI, and micro-USB sockets. The two Ethernet sockets support simultaneous connectivity with automation and information technology (IT) networks. As an open-source software platform, applications can be programmed using Node-RED, Python, and C. RevPi Connect can be upgraded with PROFINET, EtherNet/IP, EtherCAT, Modbus TCP, and Modbus RTU functionality without the use of expansion modules. Examples of RevPi Connect units include:

- PR100363, RevPi Connect S with 16 GB memory
- PR100197, RevPi digital I/O expansion module
- PR100250, RevPi analog expansion module

PROFINET and SBC PLCs

SBC PLCs can be sophisticated devices capable of supporting advanced networking protocols. Process field network (PROFINET) is an open standard for industrial networking devices like PLCs, drives, robots, diagnostic tools, etc. It runs over industrial Ethernet and is optimized for collecting data and controlling industrial equipment with real-time communications. It's available to run on most Arduino and Raspberry Pi PLCs.

Industrial automation networks need high-speed and deterministic communication. PROFINET focuses on deterministic performance that delivers messages exactly when needed and expected.

That means delivering each message with the appropriate speed based on the task being performed. Not all tasks are equally time sensitive. PROFINET can deliver messages on various protocols, including:

- PROFINET Real-Time (RT)
- PROFINET Isochronous Real-Time (IRT)
- Time Sensitive Networking (TSN)
- TCP/IP (or UDP/IP)

Conclusion

A wide range of SBC-based PLCs and industrial networking devices based on Arduino and Raspberry Pi technologies are available. They use open-source software and, in some cases, open-source hardware. Arduino PLCs are available as standard-sized units for small networks, micro PLCs for space-sensitive installations, and machine controllers for embedded applications. Quad-core Raspberry Pi-based PLCs can support more complex industrial networking applications. Raspberry Pi-based IPCs and IIoT gateways that support high levels of flexibility in network design and deployment are available.

Getting started with the Raspberry Pi Pico multicore microcontroller board using C

Written by Jacob Beningo

There is an inherent need in embedded systems to have a powerful, low-cost microcontroller unit (MCU). These devices play an important role not just in the product, but also in supporting tests, rapid prototyping, and capabilities such as machine learning (ML). However, getting started with MCUs generally requires an in-depth understanding of MCU technology and low-level programming languages. On top of that, development boards often cost between $20 and $1000, which can be too expensive for many developers. Also, it's not always the case that a development board is available, and even when they are, designers often struggle to get a board up and running.

This article introduces the Raspberry Pi Pico (SC0915) as a low-cost development board for the RP2040 MCU that provides developers with a wide range of capabilities. The article explores the Pico and some expansion boards, examines the different software development kits that the Raspberry Pi Pico supports, and demonstrates how to create a blinky LED application using the C SDK.

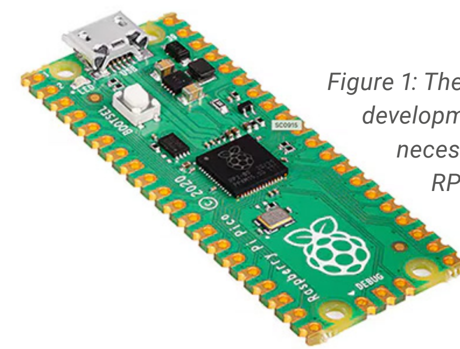


Figure 1: The Raspberry Pi Pico is a low-cost development board that contains everything necessary to develop applications on the RP2040 microcontroller. Image source: Raspberry Pi

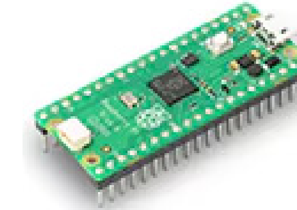
Introduction to the Raspberry Pi Pico

The Raspberry Pi Pico was first introduced in 2021 as the development platform for the RP2040 microcontroller. The Pico can be used as a standalone development board, or it can be designed into a product directly due to edge connections that can be soldered to a carrier board (Figure 1). Between the Pico's sub $5 cost and its multipurpose use, it has become a popular solution for both makers and professional developers.

The RP2040 features a dual-core Arm Cortex-M0+ processor clocked at 133 megahertz (MHz) and includes up to 264 kilobytes (Kbytes) of SRAM. The RP2040 does not include flash on-chip. Instead, the Raspberry Pi Pico provides an external 2 megabyte (Mbyte) flash chip that interfaces with the RP2040 over a quad serial peripheral interface (QSPI). The board also provides a user LED, a crystal oscillator that the phase lock loop (PLL) uses to create a stable high-speed CPU clock, and a pushbutton to configure whether the processor boots normally or into a bootloader.

Raspberry Pi Pico
- PN: SC0915
- Standard Configuration

Raspberry Pi Pico H
- PN: SC0917
- With Pins



Raspberry Pi Pico W
- PN: SC0918
- Wi-Fi

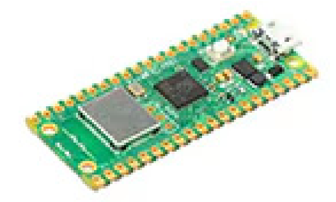


Figure 2: The Raspberry Pi Pico is available in three configurations. Image source: Beningo Embedded Group, LLC

An extensive ecosystem

The Raspberry Pi Pico already has an extensive ecosystem that allows developers to choose between using MicroPython or C software development kits to write applications for the board.

One interesting note about the Raspberry Pi Pico is that there is not just a single development board available. Instead, there are three; the original SC0915 with a standard configuration, the SC0917 which includes header connectors, and the SC0918 which includes a low-cost Wi-Fi chip for connected applications (Figure 2).

For each of these versions, the general footprint of the board remains the same. The edge connections for the board consist of 40-pin edge connections for

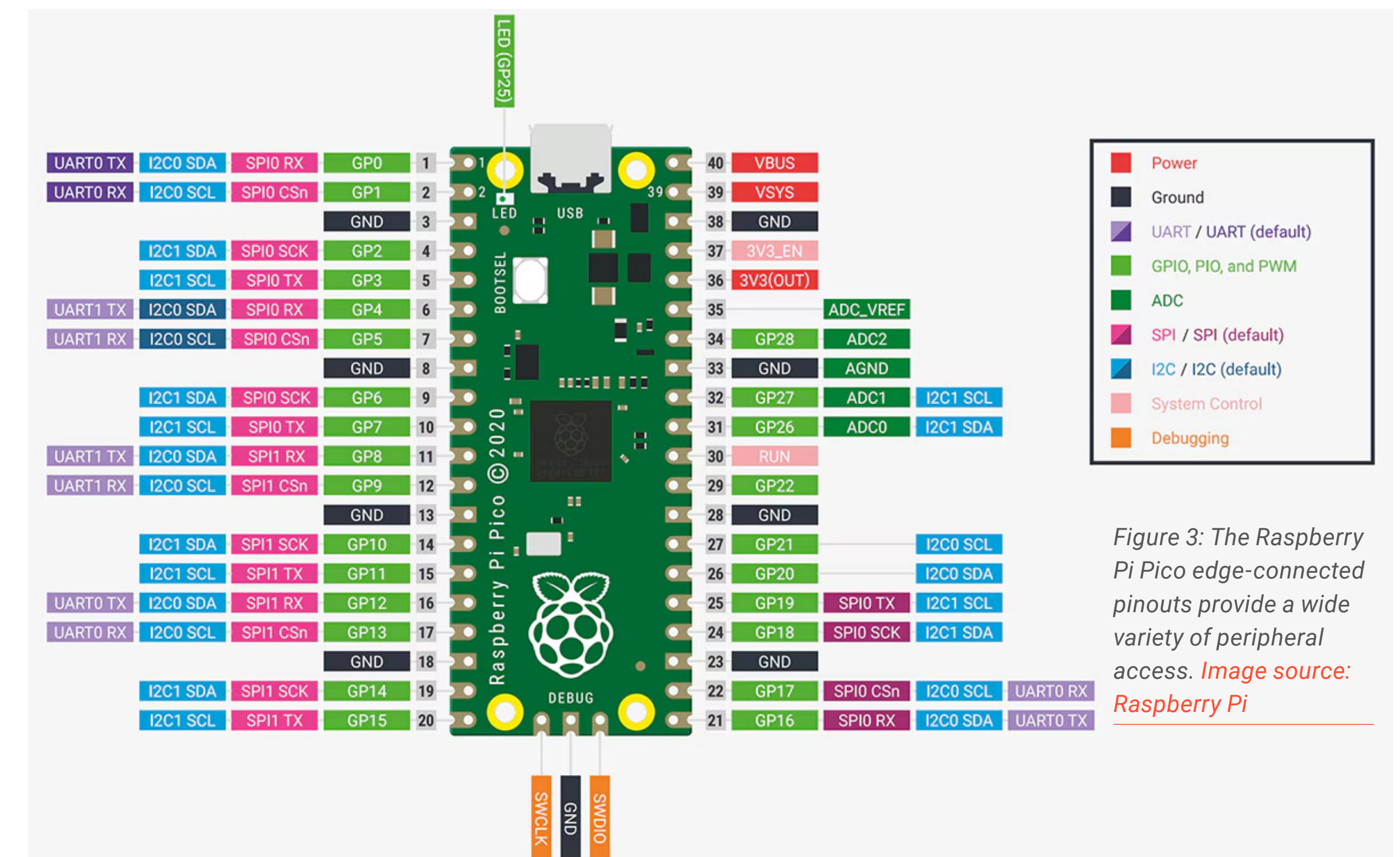


Figure 3: The Raspberry Pi Pico edge-connected pinouts provide a wide variety of peripheral access. Image source: Raspberry Pi

the peripherals and connection options shown in Figure 3. These include power, ground, a universal asynchronous receiver and transmitter (UART), general purpose input and output (GPIO), pulse width modulation (PWM), an analog-to-digital converter (ADC), a serial peripheral interconnect (SPI), an inter-integrated circuit (I2C) interface, and debugging.

Breakout board options

When the Raspberry Pi is going to be used for rapid prototyping, there is a need to gain easy access to the board's edge connectors. One option for accessing them is to populate the headers and use a breadboard. However, this solution can often result in a mess of wires that can lead to errors. So instead, there are several options for breakout boards that expand the edge connectors to more readily available interfaces.

For example, the MM2040EV Pico module board from Bridgetek breaks most of the edge connectors into pin and socket connections. Additionally, there is the 103100142 shield for the Pico from Seeed Studio that provides each peripheral interface as a connector. Each connector is pin compatible with expansion boards to add functions such as inertial sensors, motor drivers, and range finders.

To C or to MicroPython?

Embedded systems have traditionally been written in C because it balances low-level control with higher-level system application approaches. The problem with C today is that it's an antiquated, fifty-year-old programming language that is rarely taught in universities. It is also too easy to accidentally inject bugs and cause damage. Despite these potential issues, C is the language of choice for the majority of embedded systems development.

An alternative to using C, provided by the Raspberry Pi Pico ecosystem, is MicroPython. MicroPython is a CPython port designed to run on MCU-based systems. While it is undoubtedly a heavier processor user than C, it is a modern language with which many developers are familiar and comfortable. MicroPython can abstract out low-level details of the MCU and hardware. Hardware accesses are through high-level application programming interfaces (APIs) that are easy to learn – an important feature with tight project deadlines.

When selecting which software development kit (SDK) to use – C or MicroPython – developers need to focus on specific needs. Compared to MicroPython, using C will provide low-level access to the MCU's registers, have a smaller memory footprint, and be more efficient.

Setting up the C SDK

When using the C SDK to create a blinky LED application, there are several options. The first is to review the SDK documentation and follow the instructions. The second is to use a preset Docker container to automatically install all the tools necessary to get started. A third option is to install the toolchains and the Raspberry Pi Pico example code manually, including:

- Git
- Python 3
- Cmake
- gcc-arm-none-eabi \
- libnewlib-arm-none-eabi

Retrieving the Raspberry Pi Pico example code can be done by cloning Raspberry Pi's git repo using the following command:

```
git clone https://github.com/raspberrypi/pico-sdk /home/sdk/pico-sdk && \
    cd /home/sdk/pico-sdk && \
    git submodule update --init &&
```

Once these libraries and the source code are installed, the next step is to explore and compile a blinky LED application.

Writing a first blinky application

The C SDK comes with a blinky example that developers can use to build their first application. The Code Listing below uses the Pico's onboard LED and the PICO_DEFAULT_LED_PIN directive to set up an I/O pin and blink it with a 250 millisecond (ms) delay

Copy

```
/**
 * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include "pico/stdlib.h"


int main() {
#ifndef PICO_DEFAULT_LED_PIN
#warning blink example requires a board with a regular LED
#else
  const uint LED_PIN = PICO_DEFAULT_LED_PIN;
  gpio_init(LED_PIN);
  gpio_set_dir(LED_PIN, GPIO_OUT);
  while (true) {
    gpio_put(LED_PIN, 1);
    sleep_ms(250);
    gpio_put(LED_PIN, 0);
    sleep_ms(250);
  }
#endif
}
```

Code listing: the Raspberry Pi Pico uses the PICO_DEFAULT_LED_PIN directive to set up an I/O pin and blink it with a 250 ms delay. Code source: Raspberry Pi

Per the listing, the LED_PIN is assigned the default pin; calls are then made to the C gpio APIs. gpio_init is used to initialize the pin, while gpio_set_dir is used to set the LED_PIN to an output. An infinite loop is then created that toggles the state of the LED every 250 ms.

Compiling the application is relatively straightforward. First, a developer needs to create a build directory in their Raspberry Pi Pico folder using the following commands:

```
mkdir build
cd build
```

Next, cmake needs to be prepared for the build by executing the following command:

```
cmake
```

Now, a developer can change to the blinky directory and run make:

```
cd blink
make
```

The output from the build process will be a blinky.uf2 file. The compiled program can be loaded on the Raspberry Pi Pico by holding down the BOOTSEL pin and powering up the board. The RP2 will then appear as a mass storage device. The developer needs to drag the blinky.uf2 file to the drive, at which point the bootloader will install the application. Once completed, the LED should begin blinking.

Conclusion

The Raspberry Pi Pico is an attractive solution for embedded developers looking for flexibility in their development cycle. Several options are available, including standalone solutions or boards with wireless connectivity. In addition, the ecosystem supports C and C++, as well as MicroPython. Developers can pick which language works best for their application and then leverage the corresponding SDK to accelerate software development.

A guide for the ESP32 microcontroller series

Written by Don Wilcher

The ESP32 microcontrollers have spawned into a central part of the Internet of Things (IoT) and embedded controller arena. Espressif Systems, the manufacturer of the ESP32 ecosystem, has created powerful and affordable System-on-Chip (SoC) devices that integrate Wi-Fi, Bluetooth, and central processing units (CPUs) into one microcontroller package, allowing these SoCs to be ideal for embedded controllers and IoT projects.

Navigating the terrain of the various ESP32 hardware platforms and software packages can be a daunting task for the engineer or maker. This guide will provide information on technical specifications, development kits, and software design kits (SDKs) related to the ESP32 microcontroller ecosystem. With such a guide, you will be able to select the appropriate ESP32 microcontroller device for your specific project requirements.

ESP32 overview

With low manufacturing development costs and a highly effective processor, you can deploy the ESP32 to various IoT and controller projects. Some key features of the ESP32 microcontroller include Wi-Fi, Bluetooth, and a small form factor. In general, here is a brief overview of the ESP32 microcontroller.

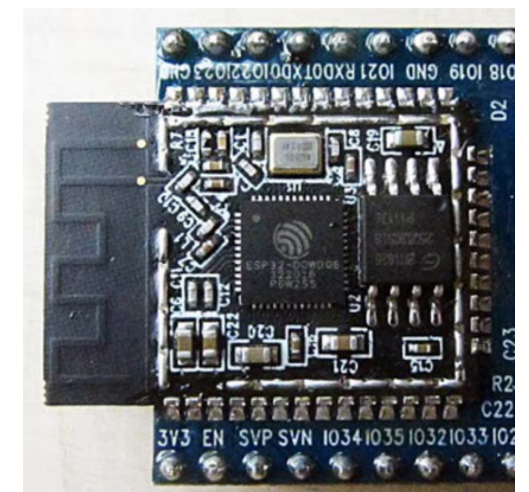


Figure 1: Typical ESP32 microcontroller. Image courtesy of [Wikipedia](https://www.wikipedia.org)

Robust design

The ESP32 microcontroller is equipped with the ability to remove external circuit interfaces dynamically. This feature ensures its reliable operation even in industrial settings. The operating temperature range of the ESP32 is -40°C to +125°C. The microcontroller can be powered by a supply voltage of +3.3V, which makes it possible to develop wireless remote sensing and controller applications.

Ultra-low power consumption

The ESP32 was designed for portable devices, wearable electronics, smart controllers, and IoT applications. Using a variety of proprietary software packages, ultra-low-power

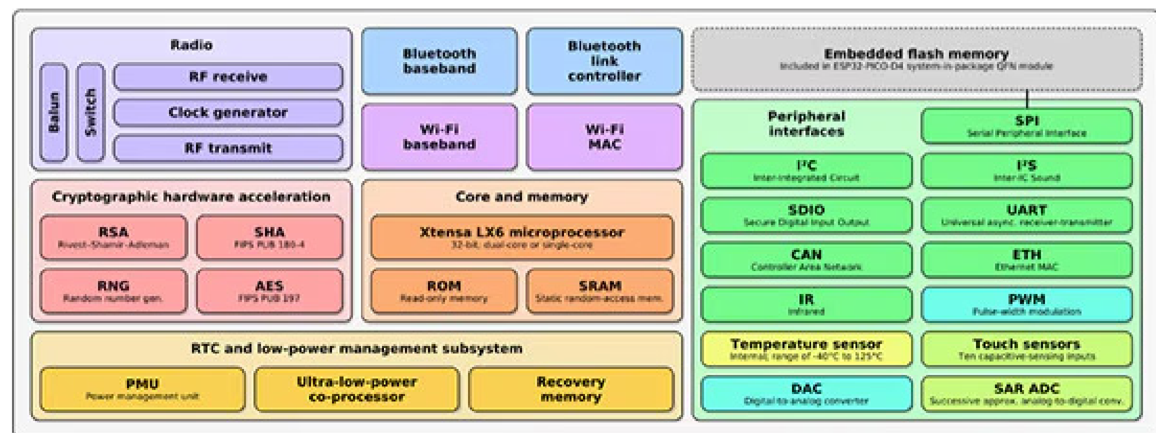


Figure 2: The ESP32 Functional Block Diagram. Image courtesy of Brian Krent (talk · contribs), CC0, via Wikimedia Commons

consumption is achieved by the ESP32 microcontroller. Additionally, the ESP32 chip has various power modes, dynamic power scaling, and clock-gating features.

High level of integration

A high level of SoC integration allows various electronic circuits to be included with the ESP32. The SoC high-level integration includes the following onboard electronic circuits.

- Built-in antenna switches
- RF balun
- Power amplifier
- Low noise receiver amplifier
- Filters, and
- Power management modules

With such features, functionality, and versatility, the ESP32 microcontroller ecosystem can provide minimal printed circuit board (PCB) space requirements to embedded applications.

Hybrid Wi-Fi, Bluetooth, and hardwired communication interfaces

You can set up a wireless system or device that can be controlled by a host controller using Wi-Fi and Bluetooth. This can help reduce the complexity of the communication system and overhead on the main controller CPU.

The ESP32 microcontrollers come with different communication interfaces like SPI, SDIO, and I2C/UART. These specialized hardwired interfaces provide other communication schemes for a host controller to control device system architecture.

The ESP32 architecture

The ESP32 Architecture is based on the Xtensa LXn CPU cores. The Xtensa CPU cores use a modular, flexible 32-bit Reduced Instruction Set Computer (RISC) architecture. A RISC device is a microprocessor architecture that uses a small effective set of programming instructions. The small set of programming instructions aids RISC architectures Xtensa processor to scale from a small cache-less controller to a high-performance digital signal processor (DSP).

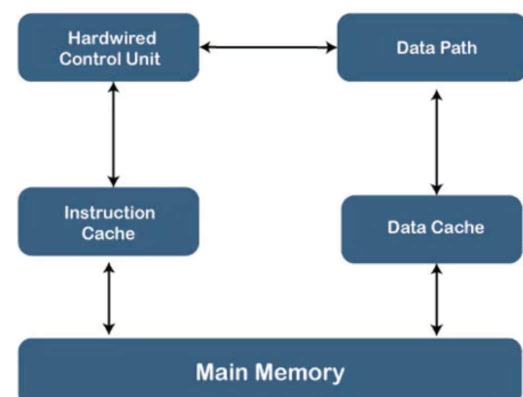


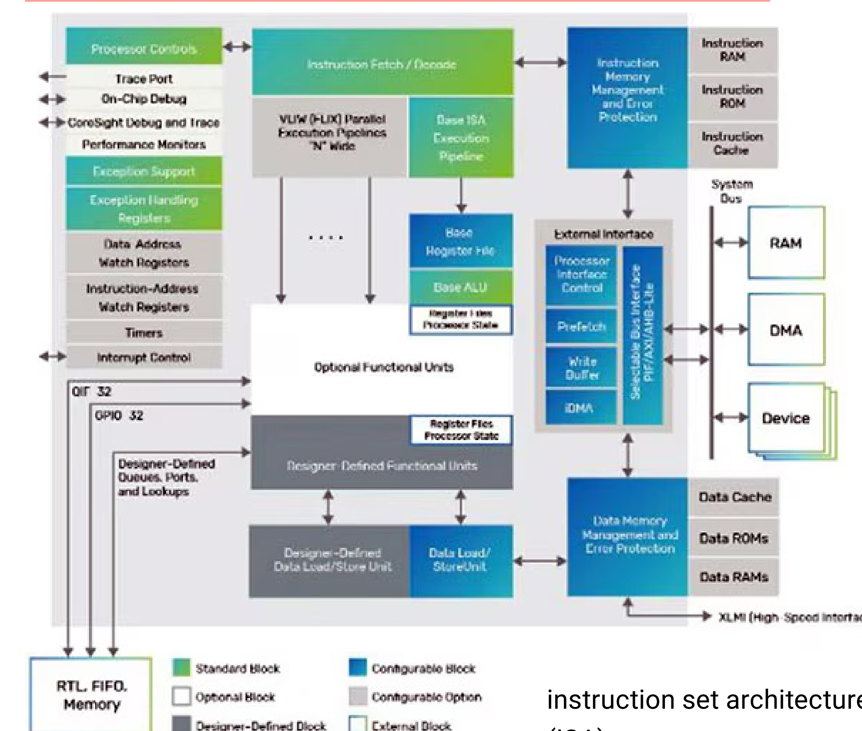
Figure 3: A typical RISC Architecture. Image courtesy of javatpoint.

The Xtensa LXn CPUs

As presented earlier, the Xtensa LXn has various CPU processing capabilities. Here is a list of some of the Xtensa LXn CPUs available for ESP32 microcontrollers:

- LX6 – the Xtensa LX6 CPU is used in the original ESP32 and varieties of the ESP32-S microcontroller family. The Xtensa LX6 is a 32-bit low-power microprocessor having dual-core and single-core configurations. The Xtensa LX6 CPU provides performance and energy efficiency for the ESP32 and ESP32-S microcontroller variants
- LX7 – the LX7 processor is an enhanced version of the Xtensa LX6 CPU. An efficient 32-bit processor architecture powers the LX7 device. Configurable RISC, data caches, and local memories are integrated into the LX7's silicon. The ESP32-S2 and the ESP32-S3 microcontrollers use the LX7 features, which are enhancements to the LX6 architecture
- RISC-V cores – the ESP32-C3 and the ESP32-C6 microcontrollers use single-core 32-bit RISC-V processors. The royalty-free, open-source instruction set architecture (ISA) removes expense costs in ESP32-C3 and ESP-C6 chip manufacturing. The RISC architecture uses 5 core blocks: a hardwired control unit (HCU), instruction cache, data cache, data path, and memory. These 5 core blocks use registers, thus allowing reasonable operating speeds for the specified microcontrollers

Figure 4: The LX7 processor architecture. Image courtesy of Cadence.



ESP32 subfamilies

If you're working on an IoT, wearable, or embedded controller project, there are plenty of ESP32 microcontrollers to choose from. Below, you'll find a list of different ESP32 microcontroller subfamilies, along with their features and some examples of projects you can create with them.

ESP32 (original variant)

- Core Architecture: Xtensa LX6 (single-core or dual-core)
- Technical Specifications:

1. Clock Speed: Range (e.g., 80 MHz - 240 MHz).
2. Memory: RAM & ROM capacity range (e.g., Up to 520 KiB RAM, 4 MB Flash).
3. Has 34 programmable GPIOs, SPI, I2C, I2S, UART, ADC, Motor PWM, LED PWM
   a. Wireless Connectivity: Wi-Fi and BLE
   b. Power Management: Low-power operation with various sleep modes
   c. Security Features: Hardware-based security (e.g., secure boot, encryption)
   d. Project Example: Smart home weather station (utilizing dual core for efficient processing).

Figure 5: ESP32 DevKitM-1

ESP32-S Series (Successors)

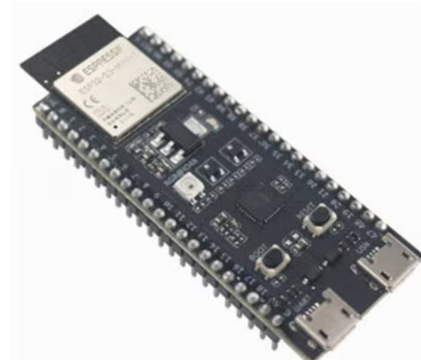
- Core Architecture: Xtensa LX7 (dual-core) - Improved performance and security.
- Variants:

1. ESP32-S2: Wi-Fi only (no Bluetooth)
2. ESP32-S3: Wi-Fi and BLE
   a. Technical Specifications (General - May vary slightly between S2 & S3)
      i. Improved clock speeds compared to the original ESP32.
   b. USB Support.
   c. Memory: 320KB SRAM, 128K ROM
   d. Has 43 programmable GPIOs, SPI, I2C, I2S, UART, ADC, LED PWM

Project examples:

- ESP32-S2: Wi-Fi smart plug (single LX7 core sufficient)
- ESP32-S3: wearable fitness tracker (dual LX7 cores for real-time processing)

Figure 6: ESP32-S3 DevKitM-1



ESP32-C series (RISC-V cores)

- Core architecture: single-core 32-bit RISC-V (potentially lower cost)
- Variants:

1. ESP32-C3: Wi-Fi and BLE
2. ESP32-C6 (Upcoming): Details not fully confirmed yet
   a. Technical specifications (General - may vary between C3 & C6)
3. Clock speed: likely similar range to other ESP32s (80 MHz - 240 MHz - Confirmation needed for C6)
4. Memory: likely similar to or increased capacity compared to ESP32-C3 (e.g., ESP32-C3: Up to 4MB Flash, 400 KiB RAM)
5. Has 14 programmable GPIOs, SPI, I2C, UART, LED PWM, ADC

- Wireless connectivity (C3): Wi-Fi and BLE (Confirmation needed for C6)

Project example (ESP32-C3): wireless soil moisture sensor (cost-sensitive application).

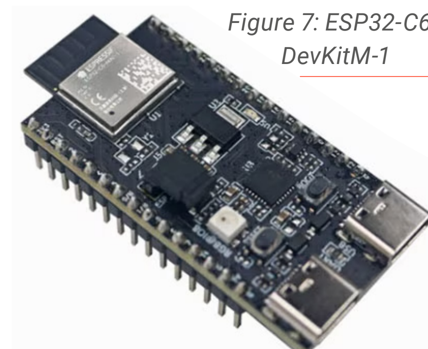


Figure 7: ESP32-C6 DevKitM-1

Figure 8: ESP32-H2 DevKitM-1

ESP32-H2 (integrates the IEEE 802.15.4 connectivity with Bluetooth 5 Low Energy (LE).

- Core architecture: single-core, 32-bit RISC-V microcontroller
- Variants: No variants as of today
- Technical specifications (general)
  1. Clock Speed -96 MHz
  2. Memory: 320 KB of SRAM with 16KB of Cache, 128KB of ROM, and Flash Memory of 4MB
  3. Has 19 programmable GPIOs with support for ADC
  4. SPI, UART, I2C, I2S, GDMA, and LED PWM
- Wireless connectivity: IEEE 802.15.4 (Mesh Network) and Bluetooth 5 (LE)

Project example (ESP32-H): smart agriculture system (can monitor environmental conditions like soil, temperature, and light levels application).

ESP32-P4 (powered by a dual-core RISCV CPU)

- Has AI instruction extensions
- Advanced memory subsystem and integrated high-speed peripherals
- Positioned for the forthcoming era of embedded applications
- Specific application areas:
  1. Human Machine Interfaces (HMI)
  2. Edge computing
  3. Increased IO-connectivity demands
- ESP32-P4 development kits coming soon

ESP32 Software Development Kits (SDKs)

With various ESP32 development kits, Espressif has provided a resource of programming tools. The SDKs are available for the ESP32 microcontrollers presented in this guide by downloading from open-source GitHub repositories. Documentation is available with sample code to ensure the main features of each ESP32 microcontroller can be easily explored. The following is a short list of available SDKs for the ESP32 microcontroller ecosystem.

- ESP-IDF – the official IoT Development Framework for the ESP32, ESP32-S, ESP32-C, and ESP32-H family of SoCs. The SDK allows typical or generic applications to be built on these microcontroller platforms. Traditional programming languages like C and C++ are used to develop microcontroller applications using the SDK
- ESP-Matter – this software implementation of the Matter protocol is a collaborative effort of the Computer Software Assurance (CSA) and company members. This consortium of company members and the CSA allows device implementation on Android and iOS controllers. The ESP microcontrollers play an integral role in the open-source Matter SDK development
- Arduino-ESP SDK – this SDK, also known as the Arduino core, is a software development kit for the ESP32 that allows developers to program this microcontroller ecosystem. The core is included with the Arduino integrated development environment (IDE). Lastly, the Arduino IDE is a collection of software libraries, and example code for the ESP32 development kits and boards

The ESP32 microcontrollers offer versatility and powerful solutions for industrial and commercial IoT applications.

Conclusion

The ESP32 microcontrollers offer versatility and powerful solutions for industrial and commercial IoT applications. The ESP32 ecosystem is integrating powerful processing capabilities, reliable connectivity, and advanced security features. Whether deployed in energy-efficient sensors or rugged industrial monitoring systems, the ESP32 microcontrollers provide the flexibility and performance needed to drive innovation and address the diverse challenges of the IoT landscape. This guide provides an overview of the ESP32 microcontroller, including its programmable platform, capabilities, and features to assist you in selecting the appropriate device for your innovative projects.

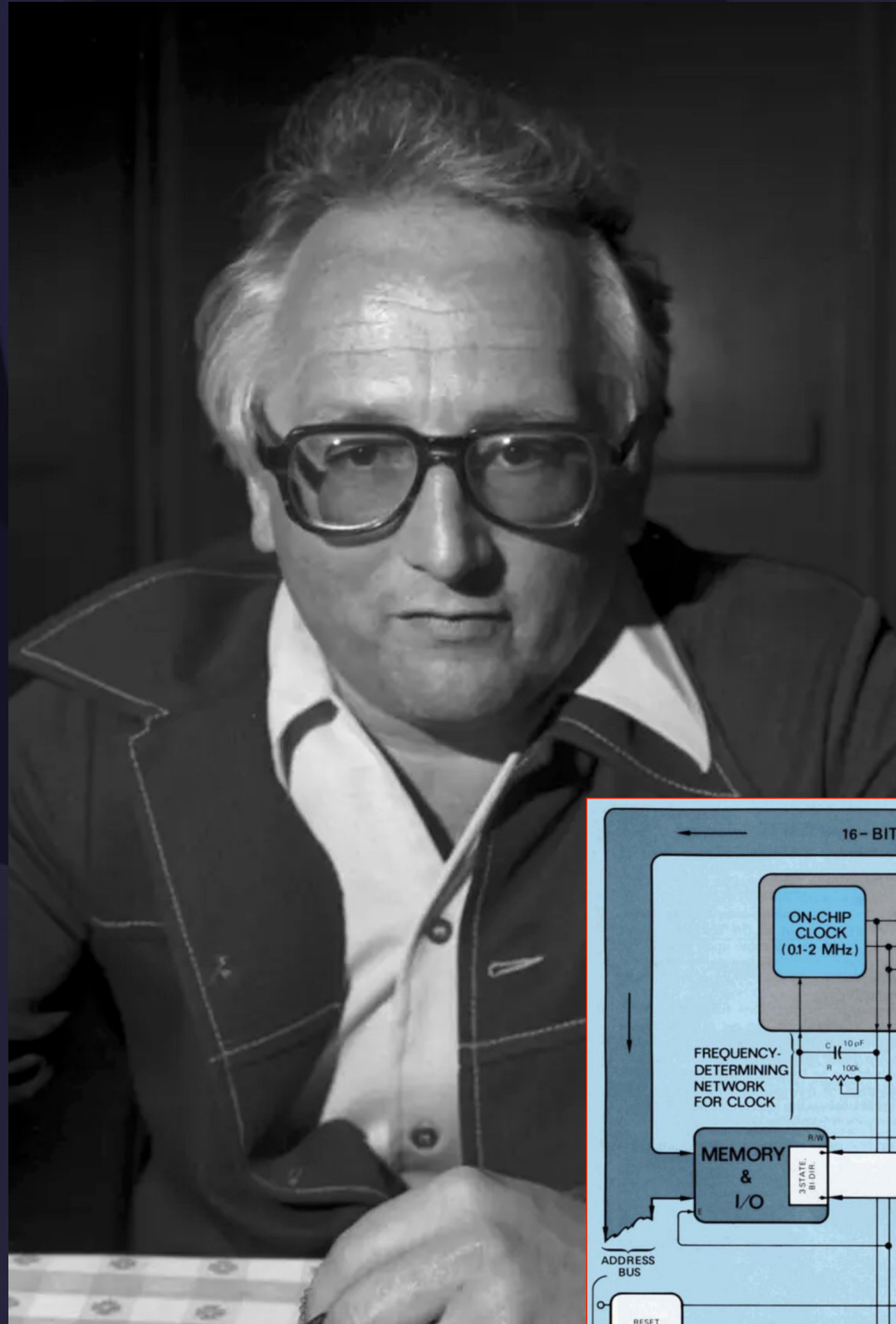


Figure 1. Chuck Peddle

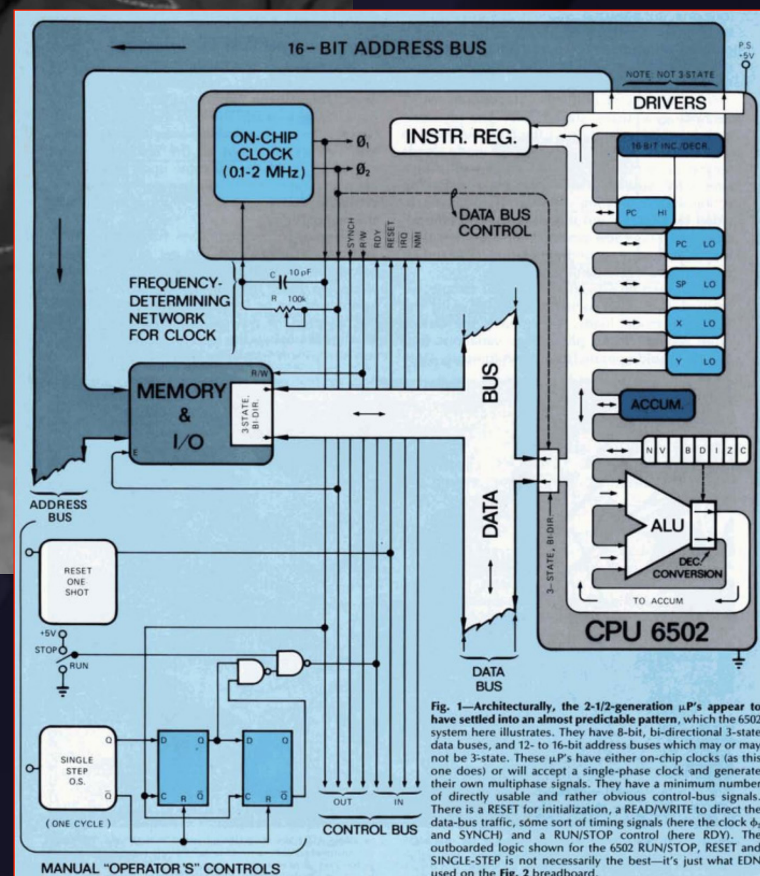


Figure 2. CPU 6502

Retro Electro: the birth of the microprocessor and Chuck Peddle

Written by David Ray, Cyber City Circuits

This is the creation story of the low-cost microprocessor market with the MOS6502. This tale is told from the perspective of Chuck Peddle, a farm boy with ambitions to never work on a farm again. While you do not see his name in lights, chances are the lights you're using to read this were made possible using several of Peddle's inventions.

Early life

Born in Bangor, Maine, Chuck Peddle grew up as a country boy in a large family with six brothers and sisters. He had to get up early with the animals and work all day just to go to bed and do it again the next day. His upbringing instilled in him a strong work ethic and solid problem-solving skills. One of his first jobs in High School was at the local radio station as a DJ, where he realized that working on the radio, with the vacuum tubes and such, was more fulfilling than talking on the radio. This started a curiosity within him that turned into a drive to learn more. After high school, he joined the United States Marine Corps Reserves, where he served in a few areas, including training in an infantry unit.

Living at the dumb end of a shovel

Peddle attended the University of Maine in the 1950s. He didn't go into college with a specific career path in mind, but he understood that taking classes would make him smarter and better prepare him for the job market, so he took a lot of physics and mathematics courses. He didn't know what he would do after college, but he knew it wouldn't involve working on a farm. In interviews, he recalls that being at the 'dumb end of a shovel' motivated him to pursue a career that provided an intellectual challenge and opportunities for advancement. He feared that if he remained in Maine, working on the farm, that would be all he would ever do.

EE 21: elements of communication

In an interview, Peddle recounts a time when he had a gap in his class schedule. He discovered a course taught by a member of Claude Shannon's team at MIT. The instructor had experienced a nervous breakdown while working for Shannon, and the research was overwhelming him. He was offered a position at the University of Maine to teach a few courses each week while on sabbatical from MIT.

The teacher had a unique way of looking at communications theory in this class. He started the class by explaining that before you can learn how to communicate, you first need to know fundamentally how

> Retro Electro fun fact:
> Claude Shannon created the foundations of all digital fields of study with his master's thesis,'A Symbolic Analysis of Relay and Switching Circuits.' Learn more about Shannon and his participation in the 1956 Dartmouth College Summer Research Project in the Retro Electro Article,'Programming a Calculator to Form Concepts.'

Figure 3. University of Maine: 1958 Course Catalog

21. Elements of Communication.—Characteristics of the auditory and vocal systems; elements of image analysis and vision; colorimetry; visual and aural aspects of information transfer, information theory; coding and decoding of information; noise; storage of information; principles of feedback and automation. Prerequisite, Ps 2 and Ms 12. *Rec* 2, *Cr* 2.

It's a wonder at all that Chuck Peddle became an engineer, much less the father of personal computing. Growing up in a working-class family in Augusta, he says he never intended to go to college.

Figure 4. Chuck Peddle 1960

the eyes and ears work. This 'first principles' approach that he learned from this class made Chuck's ambitions take a hard right turn toward computers.

General Electric

After he finished school, he knew two things. He wanted to work with computers, and he wanted to live on the West Coast. He applied to lots of places and then settled upon a junior position at General Electric, who was the first business in the world to own and operate its own computer system.

One of Peddle's first positions was on the first hard disk team at GE, where he played a pivotal role in advancing storage technology. Among Peddle's early accomplishments is his patent for 'Zoned Bit Recording' while on this team. Chuck Peddle patented the pattern used with 'zoned bit recording' for hard drives. This work contributed to optimizing data storage, making hard disk drives more practical, and gave him a deep understanding of hardware design and large-scale systems. These experiences honed his technical expertise and prepared him for future challenges.

This showed the 'higher ups' that Peddle would bring a unique perspective, and that put him on different teams while at General Electric, settling on him having a desk in the networked systems department.

Online point of sale terminals

The popularization of payment cards began in the early sixties. BankAmericard and Master Charge (now Visa and Mastercard) were becoming the preferred payment method for consumers. You might

> Retro Electro fun fact: at this time, Butler Lampson at UC Berkeley was working on 'Project GENIE' for ARPA, which developed the key systems needed for effective and optimized timesharing of mainframe computers. Learn more in the Retro Electro Article 'Project ALOHANET - Task II.'

be old enough to remember when stores used the 'knuckle buster' to obtain an imprint of your card to call it in to confirm your account before you could complete your purchase. The rise in central computing and mainframes made it possible for remote terminals to connect to powerful computers on the other side of town. This could allow cash registers in a store to have a terminal for automatic credit verification. This is when the magstripe on payment cards was introduced.

They debuted the system at a local JCPenney as a full-scale test. Everything went smoothly, but on the busiest shopping day of the year, the day after Thanksgiving, the system had a catastrophic failure. The entire computer division had egg on its face. This brought to the front Peddle's new concept of a computer inside every part of the shopping experience: 'Distributed Intelligence.'

With this new concept, Peddle approached companies like Exxon, looking for cooperation in developing this new online point of sale (POS) terminal.

> "When I left my campus (at the University of Maine) in 1959, there was not a single computer on the campus."
>
> \- Peddle

He saw an opportunity to put a microprocessor in the cash register, fuel dispenser, card reader, etc. He felt like it was going in a strong direction when it abruptly ended. In 1970, General Electric sold its computer division to Honeywell, and Peddle's POS dream hit a wall.

Intelligent terminal systems

Peddle did what most unemployed inventors do, he started his own business. Intelligent Terminal Systems. For the next few years, he was determined to develop the Electronic Cash Register. With a small team, he tuned the concept and design to the limit of available technology. There simply were not enough logic chips to manage the computer needs of a gas station.

Not yet.

Motorola - We want to turn out computers like GM turns out Chevys

In the late 1960s, a company named Viatron entered the market, attempting to lease computers to anyone who wanted one. They felt like if they could be one of the first in the market, there was no way they could lose. Having your own computer stood in stark contrast to the timeshare model, which was the best way to access a computer. Numerous businesses would have paid any amount of money to have their own computer, and Viatron was offering to lease it to them for as low as $40 a month.

Investors lined up, Viatron made millions, and soon they found themselves in a real 'Elizabeth Holmes situation.' They had partnered with Motorola to design custom ICs for their product. While Motorola worked to create the chips, Viatron focused on securing as many preorders and as much investment capital as possible. This continued until the machine was supposed to be released in February 1970. Delays at Motorola postponed the product's release schedule, revealing that the product 'System 21' sort of didn't actually exist.

Tom Bennett

Viatron went bankrupt in March 1971, leaving its creditors in the wind. The Motorola team developing the chips for the 'System 21' computer suddenly didn't have anything to work on. The mythical Tom Bennett headed

> Retro Electro fun fact: this coincides with the industry-wide recession that caused Hans Camenzind to get laid off from Signetics. Learn more in the Retro Electro Article: 'Five Five Five: The Story of Interdesign Inc.'

this team, and legend goes that Bennett used the incomplete System 21 designs as a starting point to design the Motorola 6800.

The Motorola 6800

Chuck Peddle's efforts to develop the Intelligent Terminal Systems product led him to the doors of Motorola. Tom Bennett hired him to work on the 6800 project, and Peddle, along with Bill Mensch, was instrumental in completing the project and designing peripherals for the 6800.

When the 6800 was ready for the market, Peddle switched to being a Field Applications Engineer. His time working with Exxon and Intelligent Terminal Systems gave him a sharp ear to customer wants and needs. The 6800 sold for around $300 a unit in 1974, so it was important that it was used where it made the most sense, and it was Peddle's job to figure it out. This microprocessor was highly robust and capable, particularly for the early 1970s, but its complexity made it too expensive, and the team knew this.

Peddle had his own take on the

> "It's too bad we did not patent the **Expletive** out of it, because we could have been very wealthy as a result."
>
> \- Chuck Peddle on the Electronic Cash Register

history of microprocessors. He feels that the legendary Intel 4004 (1971) and 8008 (1972) aren't actually the first true microprocessors. While he appreciated the contributions of those earlier chips, he saw them more as calculator chips. Instead, Chuck believes the title of the first real microprocessor belongs to Tom Bennett's 8-bit Motorola 6800 (1974).

Armed with these customer surveys, the team started working on another microprocessor, one far less capable than the 6800 but an order of magnitude cheaper. He found that the customer needs were simple and didn't require most of the instruction set the 6800 had. This low-cost, slim-down version would be an easy sale, but Motorola had a lot invested in the 6800, and it needed to be successful. Motorola shut down the project and decided to move the team to Austin, Texas.

Chuck Peddle, being a very smart businessman, took this opportunity to claim ownership of the intellectual property for their low-cost microprocessors. In an interview he tells the story as: *"So, they sent me a formal letter from the attorneys saying that 'You have to stop working and selling the concept of a low-cost computer. We are going to only do this (the 6800)'. I wrote to them and said, 'As of this moment, I will not work on any micro-processor for you again because I'm going to go do that micro-processor and you just did what is called in our industry 'product abandonment, and so now you can't patent it. You can't claim it as yours.' We then went off and found a company (to join) in Pennsylvania."*

Several of the team had little interest in moving to Texas, but still wanted to develop this low-cost concept. Of the twenty or so team members that worked on the 6800, eight jumped ship with Peddle. These eight include:

- **Chuck Peddle** – Program Manager
- **Bill Mensch** – Key Designer
- **Rod Orgill** – Layout Designer
- **Harry Bawcom** – Layout and Manufacturing Engineer
- **Wil Mathys** – Design Engineer
- **Terry Holdt** – Project Manager
- **Ray Hirt** – Design Engineer
- **Mike Janes** – Sales and Marketing

After leaving Motorola, they knew they needed to find a manufacturing partner if they were going to get off the ground with their low-cost microprocessor concept. They tried to get in with several companies and found they worked best with a small calculator chip manufacturer in Pennsylvania, MOS Technology (pronounced M-O-S Technology).

'You have to stop working and selling the concept of a low-cost computer. We are going to only do this (the 6800)'

MOS technology

MOS Technology was a pivotal player in microprocessor history. It was founded in 1969 in Valley Forge, Pennsylvania by former General Instrument executives with the original goal of producing affordable calculator chips. Its name, an acronym for Metal Oxide Semiconductor, reflected its focus on innovation in semiconductor manufacturing.

The early years were characterized by survival in a volatile market. The calculator industry faced disruption due to Texas Instruments' aggressive pricing, compelling MOS Technology to diversify its offerings. The company's trajectory shifted in 1974 when Chuck Peddle, and his team of engineers left Motorola to join MOS Technology.

Calculator wars

The team's departure from Motorola coincided with the Great Calculator Wars of the mid-seventies. Before the proper microprocessor, the market had 'calculator chips.' The electronic desk calculator wasn't just a status symbol, it was necessary for the workers of the day to keep up with the increasing pace of business.

The market was flooded with calculators. Desktop, scientific, pocket, battery-operated, etc. At the root of this market boom were the core chips sold by Texas Instruments. In the mid-1970s, Texas Instruments realized they could sell complete calculators directly to consumers and cut 'the middleman' out. Texas Instruments could manufacture a working marketable calculator cheaper than they were selling the chips to manufacturers like Commodore, Royal, Casio, Rockwell, and others.

This market upheaval, caused by TI's shrewd decisions, brought dozens of companies to their knees. Many of those companies perished, but a few overcame this by switching to computers. MOS Technology had luck getting the contract to make the custom ICs for Atari's new Pong system, which created the video game market shortly before Peddle and Co. joined.

Figure 5. Pong

The low-cost microprocessor

Going all the way back to his days at General Electric, he knew what he wanted. He understood what it would take to make this work. He had been working on this idea since before Intelligent Terminal Systems. Until now, the problem had always been that he couldn't convince the people with the money because he didn't have the market research to really show it. Nobody

Figure 6. MOS6501

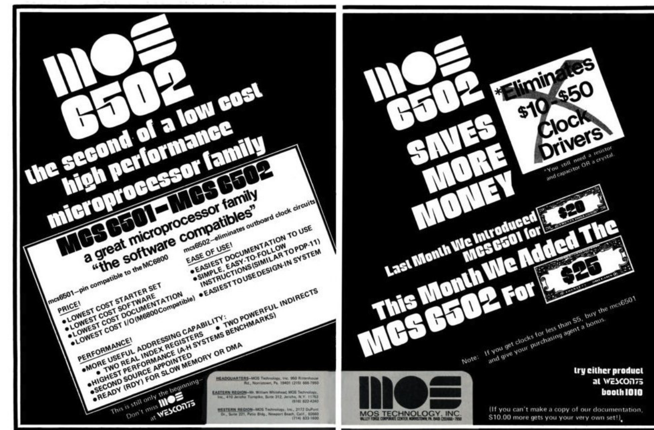


Figure 7. MOS6502

did; it was all brand new, and mainframes with terminals seemed like the most cost-effective way forward.

Once again, the team at MOS Technology did not believe in this low-cost idea, but the company's president, John Paivinen, did. Paivinen was Peddle's manager at General Electric's Computer Division in the 1960s. He would have remembered Peddle's 'Distributed Intelligence' concept, and he knew that the technology was there to make it possible now.

Starting from scratch, with their own knowledge and unique skills, they now have an in-house IC fabricator. Peddle took the billet of 'program manager' and modeled the product after all of the customer interactions he had going all the way back to the JCPenney computer crash a decade earlier. All of the lessons learned from working with GE, Exxon, ITS, and Motorola all overlapped into the perfect product summary that became MOS Technology's magnum opus. They worked tirelessly to create a functional and affordable microprocessor that would take the market by storm. They initially devised two designs: the MOS6501 and the MOS6502.

MOS6501

The MOS6501 was pin-compatible with the Motorola 6800. While it used an entirely different instruction set and was not a direct replacement, if you had a design built around the 6800, you could likely replace it with a 6501 and make it work with relatively light effort. … As a final insult to Motorola, it was sold for $20.

This part of the story ends with a lawsuit. Motorola sued MOS Technology over the MOS6501 and won, forcing MOS to pull it from the market. It is not known how many MOS6501s were actually sold, but estimates are below a thousand. Very few are known to exist today, and they can bring a lot of money online.

MOS6502

MOS Technology's crowning achievement is the MOS6502. With a limited instruction set, it was fully capable of meeting nearly every customer's needs, including cost. Through some real innovations in masking and die fabrication, MOS's manufacturing yield was higher than Motorola's and this was due to several innovations and clever design decisions.

Die size

Since the 6502 had a limited instruction set and fewer capabilities than the 6800, its die was significantly smaller allowing them to fit more on their three-inch silicon wafers. This meant that, even if they had a lower percentage yield than Motorola, they could still produce many more working units than Motorola or other companies for that matter.

Figure 8. The 6502 team from right to left: Sydney-Anne Holt, Michael Jaynes, Harry Bawcom, Chuck Peddle, Ray Hirt, Rod Orgill, Bill Mensch, and Wil Mathys. Seated: Terry Holdt.

Mask zapping

In IC production, a mask is an essential tool in the photolithography process. It acts as a stencil, defining the intricate patterns of the circuit that will be etched onto the wafer. The mask contains a series of opaque and transparent regions corresponding to the desired design, shining light on some areas and leaving shadows on others. The mask is placed over the wafer, coated with a light-sensitive photoresist material during the process. Ultraviolet light is then shone through the mask, transferring the pattern onto the photoresist. The exposed areas of the photoresist are chemically altered, allowing them to be selectively removed. This leaves a precise pattern on the wafer, which guides subsequent processes such as etching or doping.

Making the mask is expensive and time-consuming, but each time you use it, it could collect tiny dust particles and create new defects, becoming less and less effective. This is because if there's an error, scratch, dust, etc., on the mask, it will be transferred to every single die it is used with. This was a real nuisance for many companies, like Intel, Fairchild, and Motorola.

The team at MOS developed a unique way of fixing their masks using lasers to correct the mask from imperfections in between uses, called Mask Zapping. This dramatically increased yield to over 90%, while the competition was

battling with yields in the twenties.

Smart layout

Bill Mensch is sometimes called 'the Best Layout Guy in the World.' He and Peddle met at Motorola, where Mensch worked on the design and layout of the Peripheral Interfaces for the 6800 series of chips. Designs need revisions in the early stages. Engineers design the layout and mask, then wait for it to be fabricated. Once they have it in hand, they gather around and probe and prod to find the issues with it and try again. It can sometimes take as many as nine or ten attempts to get it right, but Mensch and his layout team got it perfect the very first time.

Each revision could cost a couple hundred thousand dollars, especially if you didn't have your own in-house fab shop, but Mensch did. By getting it right the first time, MOS Technology hit the starting line ahead of the competition, which had to invest millions to get their product right.

Lore has it that the die for the first 6502 worked 100% perfectly on their very first try. There have been challenges to this claim and Eric Schlaepfer tells this tale on YouTube.

The ringleader of the 6502

From here, MOS Technology took its product to WESCON75, where it gained the attention of companies such as Atari, Apple, and Commodore.

This isn't the end of Chuck Peddle. He had a powerful computing career up until his death in 2019. His journey from a Maine farm boy to a pioneer of the low-cost microprocessor market is a testament to ingenuity, confidence, faith, and vision. His innovations laid the foundation for modern computing, influencing industries and empowering the rise of personal computers. Peddle's impact is felt every time you power on a device or connect to the digital world – a legacy of making technology more accessible and affordable.

The 6502's cultural impact cannot be overstated when someone says it is one of civilization's greatest inventions. While most chips may come and go, often remaining unsung except in the catalog pages of long-forgotten, decades-old data books, the 6502 continues to be a cornerstone of microprocessing for the past fifty years.

The author acknowledges the help of the many historians who have cataloged this story over the years, especially Archive.org, Eric Schlaepfer, Ken Shirriff, Scene World, Team6502.org, the Computer History Museum, Clint Bridges, and others.

Suggested reading

1. Stephen Edwards and Bill Mensch – The Genesis of the 6502 Microprocessor (Interview)
2. Computer History Museum – Oral History of Check Peddle
3. Team 6502: The Story of the Team Behind the Chip that Launched a Revolution
4. Microsystems – Here Comes the PET
5. Interview with Chuck Peddle for Scene World Magazine
6. MOS MCS6500 Microcomputer Family Hardware Manual
7. Abort Retry Fail – The History of Commodore - Part 1
8. Viatron: System 21 is NOW!
9. Embedded Related – Development of the MOS Technology 6502 A Historical Perspective
10. The 6502 and the Best Layout Guy in the World

1937
Chuck Peddle was born in Bangor, Maine.

1938
Claude Shannon publishes 'A Symbolic Analysis of Relay and Switching Circuits.'

1950s
Attended the University of Maine, taking physics and mathematics courses without a specific career path but determined to avoid farm work.

1960
Began working at General Electric, contributing to the development of early hard disk technology and filing a patent for "Zoned Bit Recording."

1970
General Electric sells its computer division to Honeywell.

Peddle starts Intelligent Terminal Systems to develop an online payment system.

1973
Joined Motorola, contributing to the development of the 6800 microprocessor.

1974
Motorola released the 6800, priced at $300. Peddle pushed for a cheaper, more accessible processor but faced resistance from Motorola's leadership.

1974
Motorola planned to move the microprocessor team to Austin, Texas. Peddle and several team members left the company to join MOS Technology in Valley Forge, Pennsylvania.

1975
MOS Technology released the 6501 for $20 and the 6502 for $25.

Motorola sued MOS Technology over the 6501's design. As a result, it was pulled from the market.

How to select and use an audio codec and microcontroller for embedded audio feedback files

Written by Jacob Beningo

There is a growing need among embedded systems to provide high-fidelity audio instead of buzzers for user feedback, including alarms and alerts. While beeps and chirps have been effective in the past, users are expecting advanced sounds that can only be produced through playing audio from file formats such as MP3s. The problem is that audio playback can appear intimidating and add additional cost and complexity to a system. The first instinct is to find a microcontroller that can play MP3s, but this often adds several dollars to the bill of materials (BOM) and considerable complexity to the embedded software.

One solution that is particularly good at balancing the additional cost and software complexity is to use an audio codec. Audio codecs not only accept an audio data stream from a microcontroller, they often also have multiple features that allow the developer to carefully tune the audio playback system to improve the quality of the sound played by the system.

This article will discuss the role of audio codecs, the main characteristics that developers should consider when making a selection, and how to apply them effectively. Solutions from AKM Semiconductor, Texas Instruments, and Maxim Integrated will be introduced and used as examples here, though others are also available. It will conclude with tips and tricks on how to accelerate audio playback application development using a codec, while lowering system cost.

What are audio codecs?

An audio codec is a hardware component that is capable of encoding or decoding a digital data stream containing audio information1. An audio codec is useful because it allows the audio processing to be offboarded from the microcontroller. This can significantly decrease the software complexity and also allow a less expensive and less capable microcontroller to be used for an application.

A typical audio codec will contain several functional blocks:

- An I2S interface to transmit or receive encoded digital audio data
- An I2C interface to configure and read the audio codec's control registers
- A microphone input which is connected to an analog-to-digital converter (ADC)
- At least one audio output channel such as a speaker output, but most also include a line out and may include multiple speaker outputs for stereo playback
- A digital block that contains high-pass, low-pass, notch, and equalizer filters to tune audio playbacks and recordings

An example audio codec that is quite popular due to its low cost and audio capabilities is the AK4637EN 24-bit audio codec from AKM Semiconductor (Figure 1). The AK4637EN has all these

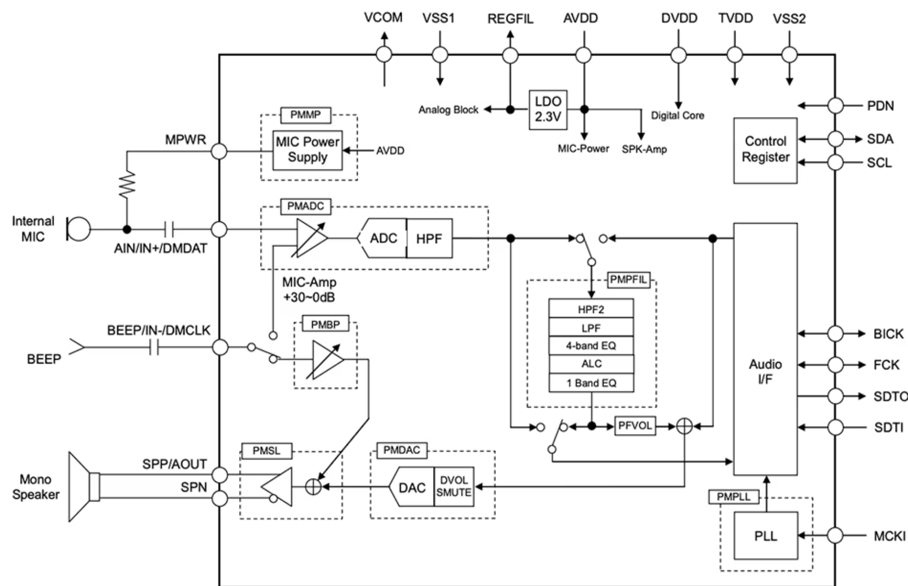


Figure 1: The AK4637EN is an audio codec with a mono speaker output that has audio playback and recording capabilities. It also contains an internal audio block that can be used to filter incoming and outgoing audio to improve audio fidelity. Image source: AKM Semiconductor

features, in addition to a beep generator input that can be used to generate a beep using a pulse width modulation (PWM) signal at a desired frequency.

Developers will find that the main differentiator for an audio codec is going to be whether it outputs mono or stereo audio, as well as the digital block capabilities. For example, the AK4637EN offers a high-pass filter, a low-pass filter, a four-band equalizer, an auto-leveling channel feature and a single-band equalizer. The latter can be used as a notch filter. How a developer sets up these digital filters can dramatically affect how a system sounds.

The audio codec can sometimes intimidate a developer that is new to audio playback. For example, while the AK4637EN is a simple audio codec, a quick examination of the datasheet shows that it has 64 configurable registers. That might seem like a lot at first, but most of those registers are used to set the filter coefficients for the various digital filters that are available. There are only a handful that need to be used to get the system outputting audio properly, making the driver development for an audio codec far simpler than a newbie might imagine.

How to select an audio codec

One of the key drivers to selecting anything in product development is cost, and audio codecs are no different. Still, it is important to keep in mind that developers get what they pay for, so when it comes to audio, a team must carefully

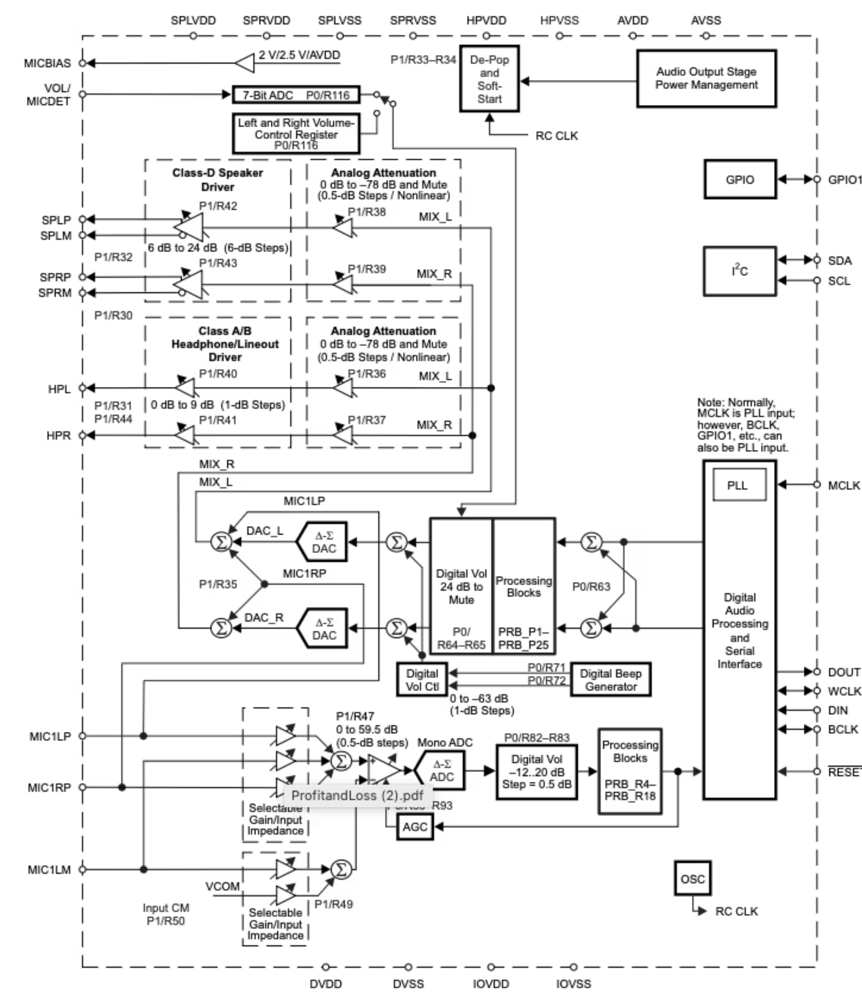


Figure 2: The TI TLV320AIC3110IRHBR is an audio codec with stereo output and amplification in addition to a microphone input. The codec can drive 1.29 watts from internal amplifiers and has programmable digital audio blocks. Image source: Texas Instruments

weigh the design requirements against the key solution parameters.

The first consideration is the required output from the audio codec. There are several different choices. For example, the AK4637EN has a line output and a mono speaker output. There are other codecs like the Texas Instruments TLV320AIC3110IRHBR stereo audio codec that can drive two speakers at 1.29 watts (Figure 2).

Other audio codecs like the Maxim Integrated MAX9867 are designed to only drive a pair of headphones (Figure 3). The MAX9867 has the typical I2S and I2C digital interfaces, but it also contains stereo microphone inputs and two line ins that can be digitally selected.

Deciding between these three solutions as to what the output type will be (or even the input) is a critical early decision.

Developers also need to consider what they will be driving. Will the audio codec be directly driving headphones, one speaker or a pair of speakers, and what will the output rating be? If the system will be driving a 5-watt speaker, there are not many codecs for embedded systems that will do that. Instead, a developer may want to select the line out and use a separate Class-D amplifier to drive the speaker directly. This saves some cost while also providing design flexibility.

Two final considerations are the internal routing and digital filtering capabilities. Here is where the real differentiation and cost differences are determined for an audio codec. For example, the TLV320AIC311IRHBR has de-pop and soft start capabilities to minimize speaker popping and allow for a smooth transition into audio playback. It also has an internal mixer for each output channel and digital volume control.

It is up to the developer to carefully balance their needs from the audio codec with the BOM and the amount of board space that will be consumed by the circuitry.

The audio playback system

When working with an audio codec, it is important to realize that there are several different blocks outside the audio codec that are necessary to achieve successful audio playback. The exact blocks will vary slightly based on application

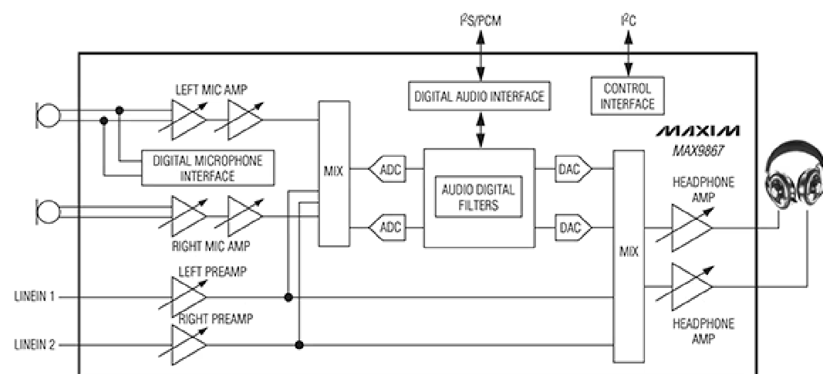


Figure 3: The Maxim Integrated MAX9867 audio codec can drive stereo headphones and select between digital, microphone and line inputs. Image source: Maxim Integrated

and the method decided on for playback, but a generalized diagram is shown in Figure 4.

There are several points in this diagram that are worth discussing. First, there needs to be some method that is used to store the audio playback files. There are two options for this; store the files internally in the microcontroller flash memory or store them externally in flash memory. The choice will depend on how large the audio file(s) are and how large the internal flash memory is on the microcontroller.

Developers also need to consider what the audio playback format will be. The most common is to use an MP3. In this case, the selected microcontroller needs to have a software stack that supports MP3 decoding. This allows the MP3 file to be opened and then pushed using a dynamic memory access (DMA) controller out via the I2S interface. Even the I2S port can be configured for master/slave and several other modes, so this needs to be carefully examined to ensure that the data is transferred to the codec at the correct rate.

As mentioned earlier, an external audio amplifier may or may not be needed depending on the application. A typical codec outputs around 1 to 1.5 watts, which is useful to drive a small speaker. To drive a 3 watt or larger speaker, it will be necessary to use external amplifier. Again, the most widely used are Class-D. The amplifier does not necessarily need to have variable gain either. The audio codec can adjust volume control digitally to provide a wide range of output power.

One area that is often overlooked is bulk capacitance. When audio is playing, it can pull heavily on the power rails. If there is not enough capacitance on the board, the output quality can be dramatically affected and can take on a twangy sound along with several other unwanted noises. This can be detected by carefully monitoring the power rails during testing. It is not a bad idea during pc board development to leave extra footprints on the board to allow different capacitance values to be tried in order to tune the output circuitry.

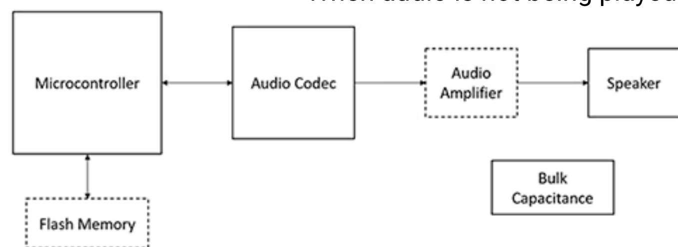


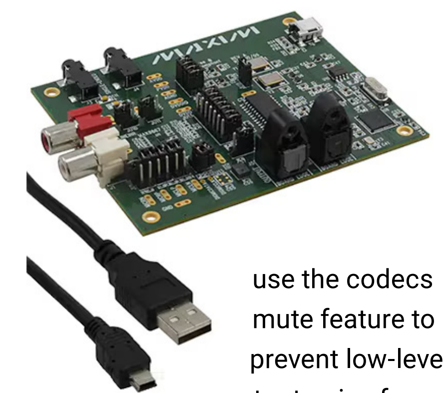
Figure 4: A generalized connection block diagram for an audio playback system in a typical embedded application shows that there needs to be storage for audio files, which can be on the microcontroller or on external memory. Image source: Beningo Embedded Group

Tips and tricks for selecting and using an audio codec

Audio codecs can dramatically simplify the embedded software and provide an application with great sounding audio quality. Audio codecs can be tricky if a developer has not worked with them before. To successfully leverage an audio codec, there are several 'tips and tricks' teams should keep in mind such as:

- Use the direct memory access controller (DMA) feature within a microcontroller to feed the audio codec with minimal CPU intervention. This will help to ensure that the codec is not 'starved' for data
- When audio is not being played, use the codecs mute feature to prevent low-level output noise from reaching the speaker
- When disabling or enabling playback, use an audio codec's soft mute feature to prevent speaker popping and other unwanted noise
- Use a terminal application to output the codec registers after the codec has been initialized. This can be especially useful when attempting to debug issues or tune the speaker output circuitry and enclosure
- Leverage the internal digital filter mechanisms included in a codec. The digital filters allow a developer to equalize the output, filter out unwanted high and low frequencies, and maximize the quality of the sound system
- Do not forget that tuning the sound will only be a useful endeavor when the circuit board and speaker are installed in the enclosure, as [the enclosure and mounting make a huge difference](#)

Figure 5: The MAX9867EVKIT+ eval kit for the MAX9867 connects to a PC over a USB cable and features RCA inputs, headphone outputs, and fiberoptic transmit and receive modules. Image source: Maxim Integrated



To get started, developers can experiment with the [MAX9867EVKIT+](#) evaluation kit for Maxim Integrated's MAX9867 (Figure 5).

The kit comprises the board and associated software and comes configured to send and receive audio data using the Sony/Philips digital interface (S/PDIF), though it can also be set to use I2S. It has two RCA input jacks, two 3.5 millimeter (mm) analog output headphone jacks, and fiberoptic receive and transmit modules. The software is Windows compatible, and when connected to a PC over a USB cable it opens into a graphical user interface (GUI) through which the developer can experiment with the MAX9867's settings (Figure 6).

Conclusion

Embedded system users have become accustomed to quality audio to the point that it is now expected instead of buzzers and beeps for alarms, alerts, and other user audio feedback. This puts the onus upon development teams to implement MP3 playback capabilities in their systems. This can at first appear to be a complex endeavor. However, by using the right audio codec alongside a microcontroller, and by following some design best practices, developers can balance the cost and complexity associated with audio applications.

References

1. https://en.wikipedia.org/wiki/Audio_codec

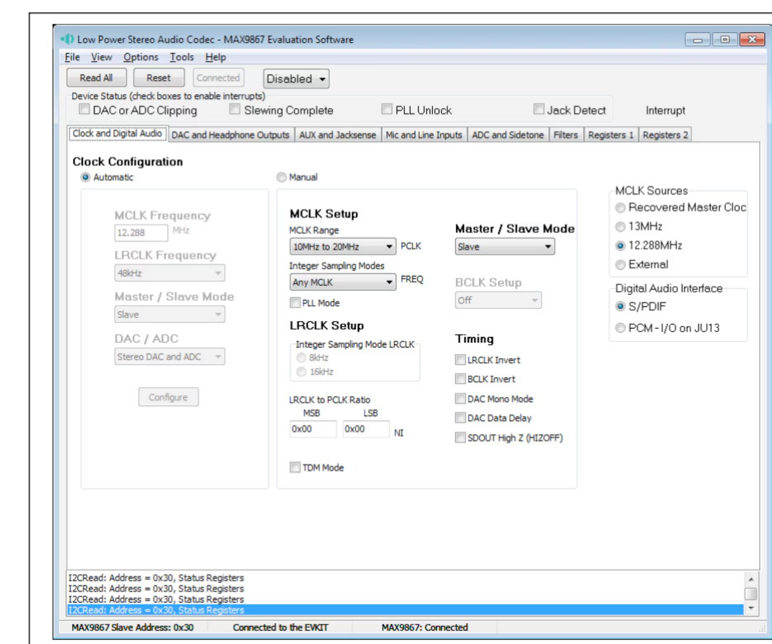


Figure 6: Using the Windows-based GUI, users can experiment with a wide range of MAX9867 settings, starting with Clock and Digital Audio (selected tab), all the way to Registers 1 and Registers 2 (right). (Image source: Maxim Integrated)

How to perform firmware updates without halting firmware execution

Written by Bill Giovino

Internet of Things (IoT) sensor-based applications are expanding, and so too is the size and complexity of the microcontroller firmware in the IoT endpoint. This firmware must become more efficient to speed execution, which is one reason flash firmware updates in the field are a necessity. However, securely updating firmware in the field usually requires halting execution of firmware while the update is in progress. Depending on the architecture, the size of the update, and network speed, this can be accomplished in as quickly as a minute or as long as an hour. For critical applications this delay can be unacceptable.

This article explains the considerations for updating interrupt-driven firmware in the field and the need to keep executing application firmware while the update is in process. It then introduces the PIC32MZ2048EFH144T-I/PH microcontroller from Microchip Technology and shows how it can be used to execute firmware while simultaneously receiving updated firmware over a network.

The importance of firmware updates

Firmware is updated for four main reasons: to correct bugs in the code, to add or improve features, to make adjustments to system security, and to make firmware more efficient. Code efficiency is measured by the number of clock cycles it takes to perform a specific task or thread. The fewer clock cycles to perform a task, the more efficient the code, which speeds execution and usually (not always) reduces code size. This is especially true for IoT sensor-based endpoints as these applications are interrupt-driven and so must quickly switch context whenever a sensor or peripheral generates an interrupt.

Two factors affecting the efficiency of interrupt-driven IoT applications are the efficiency of the architecture, and the efficiency of the code. While it is impractical to change the architecture of a microcontroller in the field, it is practical and normal to update the microcontroller firmware to improve efficiency.

Sensor-based firmware is almost always interrupt-driven. Intelligent sensors connected to a microcontroller serial port can generate an interrupt to the microcontroller to halt normal execution so the firmware can vector to an interrupt service routine for that particular sensor. This is more efficient than sensors that need to be periodically polled to determine if the sensor reading has new data to transmit. The advantage of an interrupt-driven sensor strategy is that the microcontroller only spends clock cycles on reading the sensor when there is useful data to receive.

Polling wastes clock cycles when firmware has to access the sensor to read data that is discarded because the sensor reading has not updated.

With multiple sensors and tasks comes multiple subroutines and interrupt routines to manage them, increasing code size. Complex code requires some form of real-time operating system (RTOS) to manage all these separate tasks. The RTOS can be a simple firmware application written by the software engineer or an off-the-shelf product. The RTOS manages the different firmware tasks to make sure each individual task starts and finishes within the time necessary for the application to operate properly. If many tasks need to be managed by the RTOS, it is beneficial for the application for tasks to finish in as few clock cycles as possible. This prevents different tasks from delaying each other.

When an interrupt is received, the time it takes to complete the interrupt service routine is mostly a combination of three factors:

1. The clock cycles required to recognize the interrupt and begin to vector to the interrupt service routine. If the task is lower priority than the task that is running, this will be delayed until the present task is complete. This is application dependent
2. The clock cycles required to save the context of the present CPU state and vector to the interrupt service routine. This is architecture dependent and outside of the control of the software engineer
3. The clock cycles required to execute the interrupt service routine. This depends upon both the complexity and the efficiency of the code written by the software engineer

The more efficient the firmware, the less likely a conflict will occur between tasks that need to finish within a certain period.

Flash firmware update memory requirements

Systems that need to be reliably updated in the field require twice the estimated flash program memory needed for the application. This is because the flash memory must be large enough to contain both the existing flash firmware and the updated firmware. However, it is common for small systems running only from internal flash program memory to halt code execution while the firmware update is being received over the network. This can be unacceptable for mission-critical applications and is contrary to the target of efficient firmware – i.e., code that is stopped is running at zero percent efficiency!

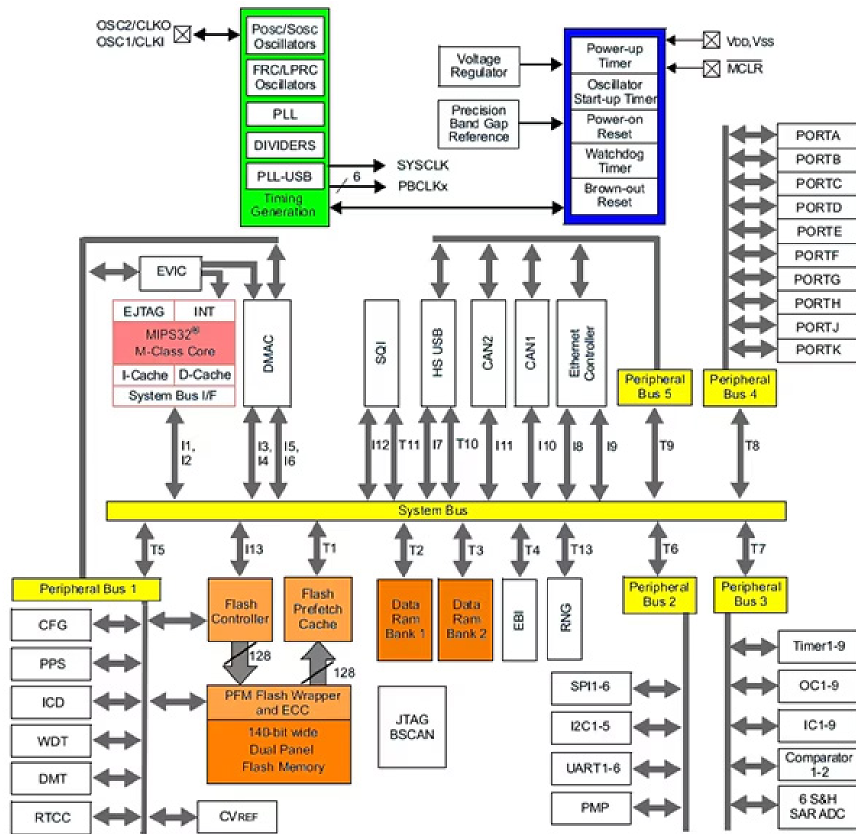
Executing firmware while updating flash

A high-performance microcontroller that can execute firmware while updating the on-chip flash memory is the Microchip Technology PIC32MZ2048EFH144T-I/PH microcontroller (Figure 1). The PIC32MZ2048EFH144T-I/PH is based on the MIPS32 M-Class core architecture with a floating point unit (FPU) that targets complex interrupt-driven IoT endpoints. It has 2 megabytes (Mbytes) of program memory flash and 512 kilobytes (Kbytes) of SRAM. It also has 160 Kbytes of boot flash. The PIC32MZ2048EFH144T-I/PH core can run as fast as 252 megahertz (MHz) over a -40°C to +85°C temperature range, and at 180 MHz over -40°C to +125°C. Operating voltage is a low 2.1 volts to 3.6 volts.

It has nine 32-bit capture/compare timers to support complex firmware as well as measuring external signals.

External serial ports include nine UARTs and five I2C ports. There are six SPI ports that also support the audio I2S interface. A 12-bit analog-to-digital converter (ADC) with 48 inputs can measure voltages from precision analog sensors. With these many serial ports and ADC inputs, the PIC32MZ2048EFH144T-I/PH can interface with many external sensors, making it appropriate for complex sensor-based IoT

Figure 1: The 252 MHz Microchip Technology PIC32MZ2048EFH144T-I/PH is based on the MIPS32 M-Class architecture and has a wide range of serial ports for interfacing to external sensors. Image source: Microchip Technology



endpoints. Two CAN 2.0b ports allow the microcontroller to network with industrial and automotive networks that use the common CAN protocol.

An Ethernet port supports 10/100Base-T networking. A USB 2.0 Hi-Speed controller supports an external interface for additional peripherals or debugging and also supports USB On-The-Go (OTG).

Each of these peripherals can generate one or more interrupts. With so many sensors and interrupt sources, maintaining code efficiency becomes a necessity.

To improve efficiency the MIPS32 M-Class CPU core has 32 32-bit general purpose registers (GPRs). This improves efficiency by reducing accesses to external memory. Besides the usual bit set and clear instructions, the M-Class also supports single-cycle bit invert instructions. This improves RTOS efficiency by increasing the efficiency of semaphore handling. The core also has a five-stage instruction pipeline that improves efficiency by minimizing memory access conflicts, resulting in more single-cycle instructions.

The MIPS32 M-Class also has seven GPR shadow register sets. This significantly improves interrupt performance and context switching by eliminating the many clock cycles required to save the GPRs on the stack. With seven shadow register sets, the core can nest interrupts and context switches seven deep before having to spend clock cycles saving the GPRs on the stack.

The PIC32MZ2048EFH144T-I/PH has two 1 Mbyte banks of program flash memory (PFM), designated PFM Bank 1 and PFM Bank 2. Each PFM has its own dedicated boot flash memory (BFM) designated BFM Bank 1 and BFM Bank 2. The BFM does not need to be updated during a PFM update. Having these two separate banks of memory has multiple advantages. For example, the microcontroller supports dual booting, so on power-up it can be configured to boot from either flash memory bank. This allows the microcontroller to support two different device configurations.

The two banks of flash also provide the added advantage of allowing firmware execution from one flash bank while updating the firmware in the other flash bank. Microchip refers to this as Live-Update, also referred to as run-time self-programming (RTSP). When RTSP is initiated in an active IoT endpoint executing firmware out of PFM Bank 1, firmware is received over the network in blocks. The recommended method for managing firmware updates over a

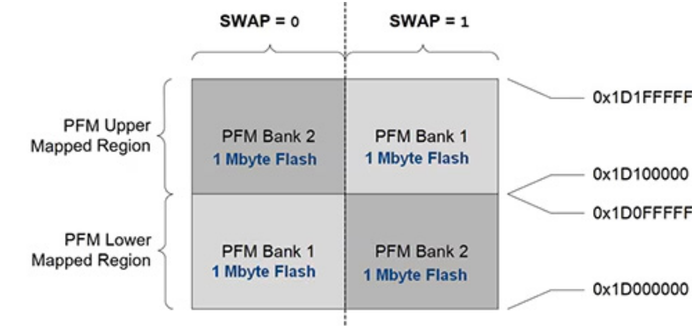


Figure 2: The PIC32MZ2048EFH144T-I/PH microcontroller has two independent banks of PFM. If SWAP=1, firmware can run out of PFM Bank 1 while PFM Bank 2 is being updated. Clearing SWAP=0 allows the microcontroller to boot out of PFM Bank 2. Image source: Microchip Technology

network is to store the block of new firmware in SRAM. After receiving a complete block, the firmware executing out of PFM Bank 1 can initiate a programming sequence of the SRAM data into PFM Bank 2. While this firmware is being programmed, firmware execution out of PFM Bank 1 can continue.

When the block programming is completed, firmware can request the next block of code over the network and the sequence repeats. This continues until the block of code in PFM Bank 2 is completed. Once the programming is complete, firmware can configure the PIC32MZ2048EFH144T-I/PH on the next reset to boot from BFM Bank 2 and execute the new firmware in PFM Bank 2 by clearing the SWAP bit in the NVMCON configuration register (Figure 2). If the PIC32MZ2048EFH144T-I/PH firmware must be updated again while SWAP=0, firmware can execute out of PFM Bank 2 while simultaneously updating PFM Bank 1.

The status of the SWAP bit can be changed from either the BFM or the PFM depending upon the needs of the firmware.

Developing dual-boot firmware

For development with the PIC32MZ2048EFH144T-I/PH microcontroller, Microchip Technology provides the DM320007 PIC32MZ starter kit (Figure 3). This board supports multiple serial ports using dedicated connectors as well as header connectors. A USB Host port is used for debugging while a USB OTG port can be used for the application. A USB-to-UART/I2C connector, when connected to a PC USB port, creates a virtual COM port on a connected host PC. This allows the host PC to communicate to the I2C port on the PIC32MZ.

A 40-pin expansion header connector allows access to additional I2C, SPI, and UART ports as well as general purpose I/O (GPIO) pins on the PIC32MZ EF. There are three pushbuttons and three LEDs that can be configured by firmware.

Conclusion

IoT sensor endpoints in critical systems are demanding higher memory requirements due to increased code complexity. The more complex the code, the more it is necessary to improve firmware efficiency in order to improve the response times of context switching in the firmware. By selecting a microcontroller that can efficiently run interrupt-driven code that can simultaneously retrieve and update firmware, developers can improve the reliability of critical IoT applications without sacrificing performance.

Figure 3: The Microchip Technology DM320007 compact starter kit supports the development and testing of USB and Ethernet applications with the PIC32MZ2048EFH144T-I/PH microcontroller. It includes connectors for USB OTG, USB Host, 10/100 Ethernet, and UART/I2C. Image source: Microchip Technology

How to implement Time Sensitive Networking to ensure deterministic communication

Written by Jeff Shepard

Deterministic communication is vital in various applications such as autonomous robotics and other Industry 4.0 systems, 5G communications, automotive advanced driver assistance systems (ADAS), and real-time streaming services. The IEEE 802 Ethernet standards, called Time Sensitive Networking (TSN), have been expanded to support deterministic communication. Properly implemented, TSN can be interoperable with non-TSN devices, but deterministic communication is only available between TSN-enabled devices. There are numerous IEEE 802 standards to coordinate when implementing TSN and ensure that it delivers both deterministic communication and interoperability, making it complex and time-consuming to design TSN into networking equipment from scratch.

Instead, designers of networking equipment can turn to microprocessor units (MPUs) with built-in TSN functionality to speed time to market and reduce development risks. This article reviews the basics of TSN operation and implementation, introduces some of the many IEEE 802.1 standards for implementing TSN, considers how IEC/IEEE 60802 relates to TSN, and compares TSN with other protocols like EtherCAT, ProfiNet, and EtherNet/IP. It then presents MPUs from Texas Instruments, NXP, and Renesas that include TSN capability, along with development platforms that support the integration of deterministic networking into Industry 4.0 devices.

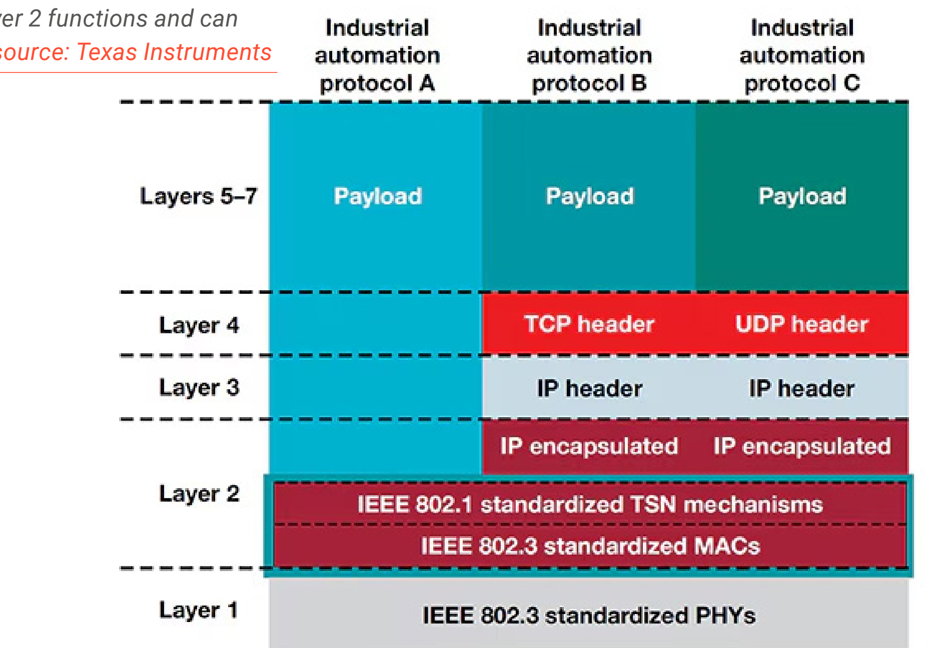


Figure 1: TSN standards define layer 2 functions and can coexist with various APIs. Image source: Texas Instruments

Prior to the development of TSN, real-time networking was only available on specialized industrial field buses. Field buses are often referred to as the 'industrial Ethernet'. The 802.1 TSN standards define layer-2 functions and local area networking (LAN) level switching and add the concepts of time and synchronization. TSN does not replace protocols at levels above layer-2 and does not define the software interface or hardware configurations and features, making it compatible with various application programming interfaces (APIs) (Figure 1).

Existing TSN traffic shaping algorithms enable the co-existence of real-time traffic with regular best-effort traffic within standard Ethernet networks. Determinism and low latency can be guaranteed for time-critical communication. That can support the deployment of safety-related systems in industrial and automotive environments. Some of the key IEEE 802.1 TSN sub-standards include (Table 1):

- IEEE 802.1 AS – timing & synchronization
- IEEE 802.1Qbv – time-aware shaper
- IEEE 802.3Qbr – interspersed express traffic
- IEEE 802.1Qbu – frame preemption
- IEEE 802.1Qca – path control & reservation
- IEEE 802.1CB – redundancy
- IEEE 802.1 Qcc – enhancements and improvements for stream reservation
- IEEE 802.1 Qch – cyclic queuing & forwarding
- IEEE 802.1Qci – per-stream filtering and policing
- IEEE 802.1CM – time-sensitive network for fronthaul

IEEE TSN can be partitioned into four categories of sub-standards that are required to ensure the operation of TSN. Time synchronization is the bedrock to ensure the synchronization of clocks across a network. 802.1AS, also called 802.1ASrev, is the primary sub-standard related to synchronization.

Another group of sub-standards relates to bounded low latency. Support for bounded low latency is a necessary condition for achieving determinism in data transmissions and is defined with five sub-standards: 802.1Qat (credit-based shaper), 802.3Qbr (interspersed express traffic), 802.1Qbu (frame preemption), 802.1Qbv (time aware shaper (TAS)), 802.1Qav (cyclic queuing and forwarding), and 802.1Qcr (asynchronous traffic shaping).

Ultra-reliability is required to deal with faults, errors, and provide redundancy and related functions.

Table 1: TSN relies on numerous sub-standards to provide deterministic performance, redundancy, and other features in a modular fashion. Image: Texas Instruments

Standard	Alias	Description
IEEE P802.1AS/Rev 1588v2	Timing & synchronization	Provides Layer 2 time synchronization
IEEE 802.1Qbv	Time aware shaper	Runs the 8 port output queues of a bridge on a rotating schedule. Blocks all ports except one based on a time schedule in order to prevent delays during scheduled transmission
IEEE 802.3Qbr	Interspersed express traffic	Interrupts transmission of an ordinary frame to transmit an "express" frame, then resumes the ordinary
IEEE 802.1Qbu	Frame Preemption	This basically just adds 802.3Qbr to 802.1Qbv. It allows for the interruption of non time critical frames to allow time critical frames through
IEEE 802.1Qca	Path Control & Reservation	Discovers the network by collecting topology information from nodes in order to find redundant paths through the network and to ensure redundancy in the future
IEEE 802.1CB	Redundancy	Messages are copied and communicated in parallel over disjoint paths and then redundant duplicates are removed at the receiver end
IEEE P802.1Qcc	Enhancements and improvements for stream reservation	Improves existing reservation protocols by supporting more streams, configurable stream reservation classes and streams, support for Layer 3 streaming, improved description of stream characteristics
IEEE 802.1Qch	Cyclic queuing & forwarding	Collects packets according to their traffic class and forwards them in one cycle. Provides a simple way to use TSN if controlled timing is desired, but reducing latency isn't important
IEEE 802.1Qci	Per-stream filtering and policing	Filters frames on ingress ports based on arrival times, rates and bandwidth to protect against excess bandwidth usage, burst sizes as well as against faulty or malicious endpoints
IEEE 802.1CM	Time-sensitive network for fronthaul	Enables the transport of time sensitive fronthaul streams in Ethernet bridged networks – new standalone TSN base standard

Related sub-standards include: 802.1CB (frame replication and elimination), 802.1Qca (path control and reservation), 802.1qci (per-stream filtering and policing), and parts of 802.1AS and 802.1AVB (reliability for time synchronization from the timing and synchronization parts of TSN and the IEEE audio bridging standard).

There is a group of general sub-standards related to dedicated resources, APIs, and other necessary 'overhead' features including higher level planning and configuration and interoperability in heterogeneous networks. Examples of these general sub-standards include: 802.1Qat (stream reservation protocol), P802.1Acc (TSN configuration), compatibility with YANG (Yet Another Next Generation) data modelling language, and 802.1Qdd (resource allocation protocol).

The modular design of TSN enables it to be optimized for specific applications and use cases. Not every feature is needed every time. For example, 802.1AS, timing and synchronization are especially important in all factory automation uses of TSN while redundancy may be required by only a subset of automation use cases.

How does IEC/IEEE 60802 relate to TSN?

At the time of this writing, the IEC/IEEE 60802, Draft 1.4, TSN Profile for Industrial Automation is out for comment and is expected to be approved sometime in 2023. This IEC SC65C/WG18 and IEEE 802 project will define TSN profiles for industrial automation. This joint effort will include profile select features, options, configurations, defaults, protocols, and procedures of bridges, end stations, and LANs to build industrial automation networks. Like the existing IEEE 802 TSN standards, 60802 will be flexible and modular and address a range of networking scenarios.

Features	EtherCAT	[illegible]	IEEE TSN
Speed	100 Mbit	100 Mbit	100 Mbit / **1Gbit**
Cycle time	31.25us	250us (31.25us)	tbd
Deterministic	yes, only EtherCAT	yes, Netload class	yes, Qci, Qbu
Time-triggered	no	yes	yes, Qbv
Synchronized	Distributed clocks	PTCP	.1ASrev multi master
Redundancy	Yes	MRP, MRPD	.1CB per stream
IP channel	no, mailbox	yes	yes
Application interface	FMMU, SYNCM	Consumer , Provider	Tbd
Engineering	PLC master	PLC controller	.Qcc, tbd

Table 2: EtherCAT, PROFINET and TSN have similar features, but implement them in different ways. Image source: Texas Instruments

IEC/IEEE 60802 will go beyond the IEEE 802 standards and is being developed in recognition of the fact that users and vendors of interoperable bridged time-sensitive networks for industrial automation need guidelines for the selection and the use of TSN related standards and features in order to effectively deploy converged networks that simultaneously support operations technology traffic and other traffic. The release of the IEC/IEEE 60802 TSN Profile for Industrial Automation could prove to be a source of confusion, at least initially, since various field buses are often referred to as the 'industrial Ethernet'.

TSN and field buses

The use of TSN and field buses is not an either-or proposition. They are compatible, often used together and all employ concepts related to time synchronization. Yet field buses like PROFINET, EtherNet/IP, and EtherCAT, implement synchronization in different ways. PROFINET uses the precision time control protocol (PTCP). EtherCAT uses distributed clocks that employ dedicated and associated registers for synchronization.

PROFINET and EtherNet/IP include the IEEE Ethernet learning bridge as the underlying switching technology. As a result, these protocols can now adapt the extension of TAS and frame preemption to use standard TSN hardware. EtherNet/IP uses UDP packets for data exchange and is compatible with the TSN switching layer. PROFINET supports a direct layer-2 buffer model for data supported by the programmable real-time unit industrial communications subsystem (PRU-ICSS) TSN solution.

TSN is designed to support cycle times at least as low as EtherCAT and PROFINET and other industrial Ethernet protocols. When upgraded to Gigabit Ethernet, TSN is expected to exceed the performance of the other protocols. Support for deterministic traffic in EtherCAT is limited to special types of data packets. Using EtherCAT and TSN in combination can improve flexibility. For example, around synchronization, TSN adds multi-master capabilities. All three protocols provide redundancy in different ways. TSN uses a technique like the parallel redundancy protocol (PRP) and the high-availability seamless redundancy (HSR) protocol as defined in IEC 62439-3 implement zero-loss redundancy (Table 2).

TSN does not include an application layer and does not challenge field buses at the application level. For example, interconnecting machines with switches while still using EtherCAT at the machine level can create an industrial Ethernet network that includes TSN functions. A TSN-EtherCAT integrated network does not mix the technologies but defines a seamless integration to use both technologies and realize the best performance aspects of each one.

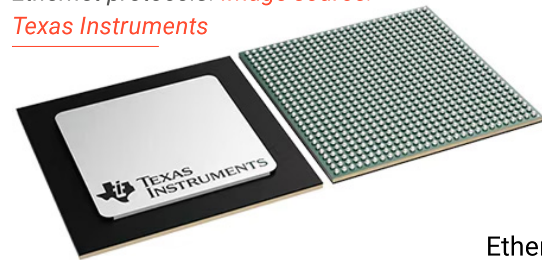


Figure 2: The AM652x Sitara processors include six ports that can be used for TSN and other industrial Ethernet protocols. Image source: Texas Instruments

MCU with up to 6 TSN ports

Designers of Industry 4.0 embedded devices that need TSN connectivity can turn to the AM652x Sitara processors from Texas Instruments like the AM6528BACDXEA. These MCUs combine two Arm Cortex-A53 cores with a dual Cortex-R5F and three programmable real-time unit and industrial communication subsystem Gigabit (PRU_ICSSG) subsystems that can be used to provide up to six ports of industrial Ethernet including TSN, PROFINET, EtherCAT, and other protocols, or they can be used for standard Gigabit Ethernet connectivity (Figure 2).

The AM652x family of MCUs includes secure boot and cryptographic acceleration in addition to granular firewalls managed by the device management and security control (DMSC) subsystem. Additionally, the dual Cortex-R5F MCU subsystem is available for general purpose use as two individual cores, or the cores can be used in lockstep for functional safety applications.

MCU with CC-Link IE TSN stack

NXP's i.MX RT1170 crossover MCUs, like the MIMXRT1176DVMAA, have a dual-core architecture with a high-performance Cortex-M7 core (running up to 1 GHz) and a power-efficient Cortex-M4 core (running up to 400 MHz). This dual-core architecture helps enable applications to run in parallel and supports power consumption optimization by turning off individual cores as necessary. These MCUs deliver a full CC-Link IE TSN communication stack and are optimized to support real-time operations and deliver a 12 ns interrupt response time.

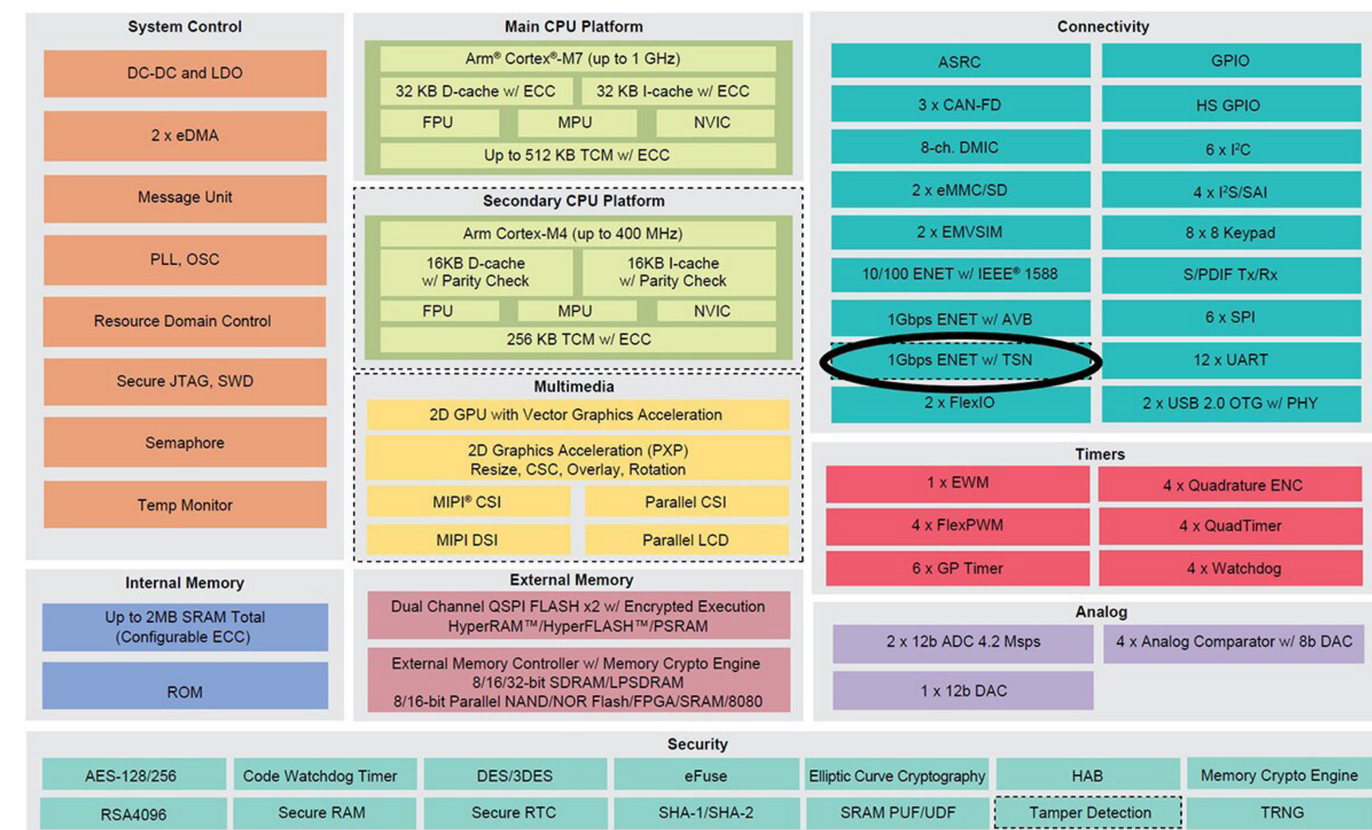


Figure 3: The i.MX RT1170 MCUs from NXP include a dedicated TSN functional block (inside the black oval). Image source: NXP

To speed the development of machine learning (ML) applications, real-time motor control, advanced human machine interfaces (HMI) like facial recognition, and other Industry 4.0 applications, NXP offers the MIMXRT1170-EVKB evaluation kit (Figure 4). This eval kit is built on a 6-layer printed circuit board (PCB) with through hole design for better electromagnetic compatibility (EMC) performance and it includes two Ethernet ports for development of TSN connectivity.

MCU and starter kit for TSN

The RZ/N2L family of MCUs, like the R9A07G084M04GBG#AC0, from Renesas are designed to simplify the implementation of industrial Ethernet and TSN in Industry 4.0 applications. They enable deterministic communications through a 3-port Gigabit Ethernet switch that supports TSN, EtherCAT, PROFINET, EtherNet/IP, and OPC UA. Renesas also offers the RTK9RZN2L0S00000BE Starter Kit+ for RZ/N2L MCUs. This starter kit includes extensive peripheral functions suitable for industrial applications and supports the evaluation of industrial Ethernet and TSN (Figure 7). The kit includes all the needed hardware and software:

- Hardware
  - CPU board with RZ/N2L MCU and on-board emulator
  - Power supply USB cable (Type C to Type C)
  - On-board emulator connection USB cable (Type A to Type Micro B)
  - PC terminal debugging USB cable (Type A to Type Mini B)
- Software
  - The development environment, sample code, and application notes are available on the web which also includes a software support package with peripheral drivers and numerous application examples for rapid evaluation and prototyping

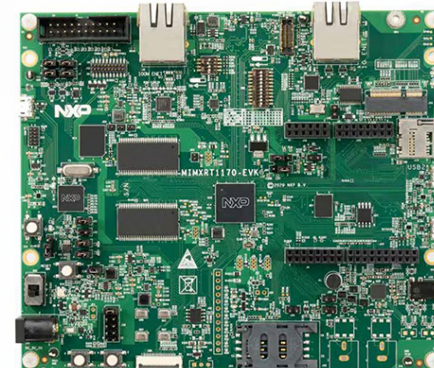


Figure 4: NXP's MIMXRT1170-EVKB evaluation kit. Image source: NXP

Figure 5: The RTK9RZN2L0S00000BE Starter Kit+ includes the necessary hardware and software, plus application examples, to support development of deterministic networking. Image source: Renesas

Summary

TSN has been added to the IEEE 802.1 Ethernet standards to support the development of deterministic communications. TSN defines layer-2 communications functions and is compatible with higher level protocols such as EtherCAT, PROFINET, EtherNet/IP, and others. It will soon be embodied in an international standard, IEC/IEEE 60802, TSN Profile for Industrial Automation. Suppliers have already begun integrating TSN into MCUs and related development platforms to help designers rapidly integrate deterministic communications into the next generation of Industry 4.0 devices.

The parts we sell help lives become richer

Imagine hearing aids that let a child experience her parents' voice clearly for the first time.

At DigiKey, the parts we sell help companies turn innovative, game-changing ideas into real-world solutions that change lives.

Find your part at digikey.com

DigiKey

we get technical

DigiKey is an authorized distributor for all supplier partners. New products added daily. DigiKey and DigiKey Electronics are registered trademarks of DigiKey Electronics in the U.S. and other countries. © 2025 DigiKey Electronics, 701 Brooks Ave. South, Thief River Falls, MN 56701, USA